# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

THE DESIGN AND IMPLEMENTATION
OF PEDAGOGICAL SOFTWARE
FOR THE MULTI-BACKEND/MULTI-LINGUAL
DATABASE SYSTEM

by

Craig W. Little

December 1987

Thesis Advisor: D. K. Hsiao

Approved for public release; distribution is unlimited

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION<br>Unclassified | 1b RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION / AVAILABILITY OF REPORT<br>Approved for public release; Distribution is Unlimited |
| 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION<br>Naval Postgraduate School<br>6b. OFFICE SYMBOL (If applicable)<br>Code 52 | 7a. NAME OF MONITORING ORGANIZATION<br>Naval Postgraduate School |
| 6c. ADDRESS (City, State, and ZIP Code)<br>Monterey, California 93943-5000 | 7b. ADDRESS (City, State, and ZIP Code)<br>Monterey, California 93943-5000 |
| 8a. NAME OF FUNDING / SPONSORING ORGANIZATION<br>8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
| 8c. ADDRESS (City, State, and ZIP Code) | 10. SOURCE OF FUNDING NUMBERS<br>PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO. |

11. TITLE (Include Security Classification)
THE DESIGN AND IMPLEMENTATION OF PEDAGOGICAL SOFTWARE FOR MULTI-BACKEND/MULTI-LINGUAL DATABASE SYSTEM (u)

12. PERSONAL AUTHOR(S)
Little, Craig W.

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM _____ TO _____ | 1987 December | 123 |

16. SUPPLEMENTARY NOTATION

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Multi-Backend Database System (MBDS), Multi-Lingual Database System (MLDS), Attribute-Based Data Language (ABDL) Interface, Relational/SQL Interfac |
| | | | |
| | | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Traditionally, courses in database systems do not use pedagogical software for the purpose of intructing the database systems, despite the progress made in modern database architecture.

In this thesis, we present a working document to assist in the instruction of a new database system, the Multi-Backend Database System (MBDS) and the Multi-Lingual Database System (MLDS). As the course of instruction describes the creation and manipulation of each language, this pedagogical aid will demonstrate, utilizing the live database system, the procedures and methods learned in class.

| 20. DISTRIBUTION / AVAILABILITY OF ABSTRACT | 21. ABSTRACT SECURITY CLASSIFICATION |
|---|---|
| ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS | Unclassified |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL<br>Prof. D. K. Hsiao | 22b. TELEPHONE (Include Area Code)<br>(408) 646-2253<br>22c. OFFICE SYMBOL<br>Code 52Hq |

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24.

#18SUBJECT TERMS (CONTINUED)

Hierarchical/DL/1 Interface, Network/CODASYL Interface.

Approved for public release; distribution is unlimited.

# The Design and Implementation of Pedagogical Software for the Multi-Backend/Multi-Lingual Database System

by

Craig W. Little
Lieutenant, United States Navy
B.A., University of Washington, 1973

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**

December 1987

## ABSTRACT

Traditionally, courses in database systems do not use pedagogical software for the purpose of instructing the database systems, despite the progress made in modern database architecture.

In this thesis, we present a working document to assist in the instruction of a new database system, the Multi-Backend Database System (MBDS) and the Multi-Lingual Database System (MLDS). As the course of instruction describes the creation and manipulation of each language, this pedagogical aid will demonstrate, utilizing the live database system, the procedures and methods learned in class.

TABLE OF CONTENTS

# I. INTRODUCTION

## A. BACKGROUND

Modern pedagogical software is essential when instructing an introductory course on database systems, utilizing a modern database system architecture. Without the supporting software, the course would be reduced to a pencil-and-paper exercise with no knowledge of the computer system software, hardware, and interface which constitute a database system. With the pedagogical aid, the student can receive the hands-on experience, on a state-of-the-art system, necessary to provide a complete learning environment.

In the Laboratory for Database Systems Research at the Naval Postgraduate School, a modern database system is operational and available to students. The database system is currently undergoing further research and development in three new directions. First, the Multi-Backend Database System (MBDS) where the database system is configured with a number of microprocessor-based processing units and their disk subsystems, called database backends. The unique characteristics of the backends are that the number of the backends is variable, the system software in all of the backends is identical, and the multiplicity of the backends is proportional to the performance gain and capacity growth of the system. Second, the Multi-Lingual Database System (MLDS) where a single database system can execute many transactions written respectively in different data languages and support many databases structured correspondingly in various data models. This allows the old transactions and existing databases to be migrated to the

new system, the user to explore the strong features of the various data languages and data models in the system, the hardware upgrade to be focused on a single system instead of a heterogeneous collection of database systems, and the database application to cover wider types of transactions and interactions in the same environment. Third, the Multi-Host Database Systems (MHDS) where a single database system can communicate with a variable number and heterogenous collection of mainframes, known as hosts, in several different data languages and allow the hosts to share the common database store and access [1].

Of the three new directions, this thesis will deal only with the development of the pedagogical software related to the user interface of the Multi-Backend Database System and the Multi-Lingual Database System.

The Multi-Backend Database System is currently configured on nine ISI (Integrated Solutions Incorporated) workstation microcomputers. One workstation functions as a controller, allowing the other eight to act as backends. Each backend has a pair of Winchester drives, one dedicated to the operating system and the other to the database system. The system is connected via an Ethernet broadcast bus and utilizes the 4.2 BSD UNIX operating system.

The Multi-Lingual Database System presently consists of a native database language, Attribute-Based Data Language (ABDL), and three different operational database languages/models, Structured English Query Language (SQL) supports the relational model, Data Language 1 (DL/1) supports the hierarchical model, and the CODASYL Data Manipulation Language (DML) supports the network model. The Multi-Lingual Database System supports these languages/models through a database

language translation and a database model transformation process into the native language.

The sample databases utilized throughout this thesis, in the various database languages, have been chosen to present a small but complete view of each database model. The queries chosen will demonstrate only the primary operations, INSERT, RETRIEVE, UPDATE, and DELETE of each database language. It is not the intention of this thesis to be an all-inclusive set of queries for the various database languages.

This thesis is to be used in support of the instruction of an introductory course on database systems, using the MBDS/MLDS system. To receive the maximum benefit from the system, the user must be familiar with the data structure, grammar, and syntax of languages supported by MLDS, prior to utilizing the system.

## B. SCOPE

This thesis expounds on the current state of development of the Multi-Backend Database System (MBDS) and the Multi-Lingual Database System (MLDS). Sample databases and queries have been developed, tested, and documented so that the documentation may be used as a lab manual. This manual provides both the instructor and students with the pedagogical aid necessary to effectively utilize the new database system to the fullest extent possible.

## C. ORGANIZATION OF THE THESIS

This thesis is divided into five chapters and an appendix. Chapter I provides a brief background on MBDS/MLDS, the importance of the thesis, and information related to the manual's composition and utilization. Chapter II and III presents an overview of the

MBDS and MLDS systems. Chapter IV describes the procedures and use of four language interfaces utilized by MBDS/MLDS. Chapter V presents summary remarks. The appendix is the MBDS/MLDS User's Manual.

## II. THE MULTI-BACKEND DATABASE SYSTEM (MBDS)

The Multi-Backend Database System (MBDS) was designed to overcome the performance problems and capacity growth problems associated with the traditional approach to database system design. These goals were achieved through the utilization of multiple backend computers connected in a parallel fashion. The backends have identical and replicated, hardware and software, and their own disk systems. In the multi-backend configuration, there is a backend controller which is responsible for supervising the execution of database transactions and for interfacing with the hosts and users. The remaining backends perform the database operations with the database stored on the disk systems of the individual backends. The controller and backends are connected by a communication bus. Users access the system through either the hosts or the controller directly.

Performance gains are realized by increasing the number of backends. If the size of the database and the size of the responses to the transactions remain constant, then MBDS produces a reciprocal decrease in the response times for the user transactions when the number of backends is inceased. This shows the direct relationship between the multiplicity of the backends and the performance gains of MBDS in terms of the response-time reduction. On the other hand, if the number of backends is increased proportionally with the increase in database size and transaction responses, then the MBDS produces invariant response times for the same transactions. This also shows the direct relationship between the multiplicity of the backends and the capacity growth of

MBDS in terms of response-time invariance. For a more detailed discussion of MBDS refer to [2-4].

## III. THE MULTI-LINGUAL DATABASE SYSTEM (MLDS)

The Multi-Lingual Database System (MLDS) supports databases structured in any of the well-known data models, such as relational, hierarchical, or network, and to execute transactions written in any of the well-known data languages, such as SQL, DL/1, or CODASYL-DML.

Since MLDS provides an environment for running database transactions written in different data languages, the transactions written in a specific data language on another database system can also be executed in MLDS. There is no need to translate a transaction written in one data language to another data language in order to run the transaction in the other database system. For example, had we wanted to run a transaction (written in DL/1 and running on IMS) in SQL/DS, we would have to translate the transaction from DL/1 to SQL, since SQL/DS is a relational system and does not run DL/1 transactions. However, in a multi-lingual database system, since both SQL and DL/1 are supported, there is no need of any translation from DL/1 to SQL. Nor is there a need of translation from SQL to DL/1. A MLDS can execute transactions written in either DL/1 or SQL.

The MLDS achieves this unique environment, to support various data models and languages, through the availability of its native data model and data language. The native data model of the MLDS is the Attribute-Based Data Model (ABDM), and the native data language is the Attribute-Based Data Language (ABDL). The difference between a conventional data model and the native data model is that all of the databases

structured in a conventional data model can be transformed into equivalent databases structured in the native model. This is known as data-model transformation. Further, all of the convential data languages can be translated into the native data language. This is known as data-language translation. It is important to note that the ABDM(ABDL) as a data model(language) is at a high level like other data models(languages), such as the relational data model (SQL data language), the hierarchical data model (DL/1 data language), and the network data model (CODASYL-DML data language) [1, 5].

## IV. PROCEDURES AND USE OF MBDS AND MLDS

### A. ATTRIBUTE-BASED DATA LANGUAGE (ABDL) INTERFACE

The ABDL interface is menu driven. We begin with a description of the five primary operations designed for the manipulation of existing databases. Then we describe how to create new databases, through the template and descriptor tables, addition of attribute values, and mass loading of records. Finally, we describe the creation and execution of queries.

#### 1. ABDL Description

The five primary operations of ABDL are INSERT, DELETE, UPDATE, RETRIEVE, and RETRIEVE COMMON. A request is a primary operation with a qualification. A qualification is used to specify the part of the database that is to be operated on. A transaction is formed by grouping together two or more requests. A description of the five types of requests follow.

The INSERT request inserts a new record into an existing database. The qualification is a list of attribute-value pairs and an optional record body. For example:

INSERT(<FILE,Cors>,<CNUM,C333>,<PLAC,Root>,<ROOM,122>)

will insert a record into the course file which includes the course number c333, the place root hall and the room number 122. There is no record body in this record.

The DELETE request removes one or more records from the database. The qualification is a query. For example:

DELETE((FILE=Stud)and(NAME=Greg))

will delete all records whose name is equal to Greg in the student file.

The UPDATE request modifies records of the database. The qualification consists of the query and the modifier. The query specifies the records to be modified in the database and is in the disjunctive-normal form. The modifier specifies how the records being modified are to be changed. For example:

UPDATE((FILE=Cors)and(CNUM=C205))<PLAC=Span>

will modify all of the records in the course file, that have a course number equal to c205 by changing the place to Spanagel.

The RETRIEVE request retrieves records of the database. The qualification consists of a query, a target-list, and an optional BY-clause. The query specifies the records to be retrieved. The target-list contains the output attributes whose values are required by the request, or it may contain an aggregate operation, i.e., AVG, SUM, COUNT, MAX, or MIN, on one or more output attribute values. The BY-clause is optional and is used to group or sort records, for example:

RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)

RETRIEVE((FILE=Stud)and(NAME>=John))(MAX(GPA))

RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)BY GPA

The first retrieve, outputs all the records, listing the names and gpa's, in the student file with names greater than or equal to John. The second, outputs only the record with the maximum gpa, listing that number. The third, outputs the same information as the first retrieve request but is sorted by gpa, beginning with the lowest number.

Finally, the RETRIEVE-COMMON request merges two files by common attribute values. The first common attribute is associated with the first retrieve and the second common attribute is associated with the second retrieve. For example:

```
RETRIEVE((FILE=Cors)and(CNUM>=C200))(CNUM,PLAC,ROOM)
COMMON(PLAC,NAME)
RETRIEVE(FILE=Stud)(GPA)
```

will find all records in the course file with a course number greater than or equal to c200 and find all records in the student file, identify records of respective files whose place and name are in common and return their course number, place, room number, and gpa. In this example, the attribute names for the common values and the target lists of the two retrieve request were different. However, they may be identical if desired.

2. Creation of The Meta Data

Now we discuss the generation of a new database in attribute-based form. After entering the ABDL interface, through the System Menu, we choose the "generate a new database" option from the Attribute-Based Menu. The Generate-Database Menu is then displayed and the "generate record template" option is chosen. A list of statements will be presented to create the database template and file templates and define the record types.

We begin by entering a UNIX file name to store the template information. To create a file template we specify (1) the database id, (2) the number of templates for the database, (3) the number of attributes for the template, (4) the name of the template, and (5) list the attribute names and value types. Items (3), (4), and (5) are repeated for each template numbered in (2). The database template is created by a collection of file templates or records of the same record type.

After we are done generating the database template, control returns to the Generate-Database Menu. The "generate descriptors" option is now chosen. We must first enter the name of the template file previously created. Then enter a UNIX file name to be used for storing descriptors. The template file must already be created prior to begin loading descriptors because for each attribute in the template database, the user is queried as to whether it is a directory attribute, and if so, what the descriptor type and descriptors will be. When using range descriptor, type A, we must be sure the ranges are mutually exclusive. MBDS does not check this. Also, when using type B or type C equality descriptor, there is only a single value and must be entered in the upper bound category. An at sign, @, is entered when the lower bound is requested, to terminate a descriptor definition for an attribute. The dollar sign, $, to terminate the entire dialogue, is automatically inserted by MBDS. Upon completion of the descriptor file, control returns to the Generate-Database Menu.

3. Generating Data Sets and Records

Now we discuss the generation of data sets and records for the new database, that will be manipulated by the attribute-based data language. We choose the "generate/modify sets" option. First we enter the name of the template file. Each attribute from each template file, will appear sequentially in the following menu:

```
CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE '___' ON TEMPLATE '___':
  (n) - generate a new set for it
  (m) - modify an existing set for it
  (s) - do nothing with it
```

The (s) option is chosen only if there are already values generated for that attribute. If the (m) option is chosen, the user may add or subtract values from the set of values that

have already been specified for a particular attribute. If the (n) option is chosen, we first enter a UNIX file name to store the set of values. We then enter the desired number of values and terminate the list of values for that attribute with an at sign, @. We are then provided with an option to modify the set previously generated. Control returns to the Generate-Database Menu, after generating the value sets.

The final operation option is now chosen, "generate records". This option allows for the generation of a large number of records, for mass loading. Again, we begin by entering the name of the template file, then a UNIX file name, to be used for storing records. Next we sequentially enter the name of the files containing the values for each attribute on the first template. The total number of records that can be generated for the first template is then presented. The total is determined by the product of the number of values in all the attribute files. The user then indicates the number of records to be generated. This is repeated for all template files. After we are done generating records, control returns to the Generate-Database Menu. The database is now complete and the quit option is chosen, returning to the Attribute-Based Menu.

4. Loading a Database

Now we discuss the loading of the newly generated database in attribute-based form. The "load a new database" option, from the Attribute-Based Menu, displays the Load-Database Menu. We begin by first choosing the "load the template and descriptor files" option. We enter the respective file names. This creates the meta data of the database in the form of various tables. Next, the "mass load a file of records" option is chosen. We enter the name of the file containing the records that were generated in the previous section. This creates the base data of the database in the form of records. Both

the tables and records are maintained by the system software. If for some reason the record file had been incorrectly assembled due to improper data in the template, descriptor, or data files, MBDS/MLDS may stop working. If this happens, refer to the RECOVER PROCEDURE at the beginning of the manual and recheck each file.

We have completed loading a new database and now exit to the Attribute-Based Menu.

5. Generating and Executing Queries

Now we discuss the request interface for attribute-based databases and the processing of ABDL requests and transactions. When the "execute the request interface" option is chosen from the Attribute-Based Menu, the Request-Interface Menu is displayed. Each of the possible options will be discussed in turn.

The option "NEW LIST; create a new list of traffic units", is chosen. Traffic Unit is MBDS terminology for a request. This option is for creating a new file of ABDL requests. The collection of menus for this option allows the user to create a file of the five primary operations. By using the menus, correct syntax is guaranteed. We begin by entering a file name to store the traffic units created during this session. Next we enter the "request" option as the desired traffic unit type. This option is chosen prior to the creation of every traffic unit and therefore, will not be repeated. The first request is an INSERT. This is the simplest of the primary operations. We enter keywords as prompted, first for the "attribute" and then for the "value". Data values must be supplied for each attribute in the record type. There is no such concept as a null value. The second, third, and fourth requests are variations of the RETRIEVE operation, a basic retrieve, a retrieve with an aggregate operation, and a retrieve with a BY-clause,

respectively. In each case, we are prompted first for the predicates of the query, an attribute, type of operator, and value, then attributes for the target-list, with or without an aggregate operator, and finally for an attribute for the optional BY-clause. The fifth request is an UPDATE. We are first prompted for the predicates necessary to build the query, same as for the retrieve request, and then the attribute and expression required to construct the modifier. The update request may only modify a single attribute at a time. If more than one attribute of a record must be modified, then multiple update requests must be used. The sixth request is a DELETE. We are only prompted for attributes, values, and relational operators as predicates for the query. The delete request is identical to the retrieve, only no target list.

The next option "SELECT" is chosen. This option is for selecting a file of previously created ABDL requests. This option presents a menu for displaying and submitting these requests for processing and allowing a new traffic unit to be executed. We begin by being prompted to use the current traffic unit file. If no, input a file name. A list of executable traffic units, numbered sequentially, are displayed to the user. The user now has four options. First, execute a traffic unit by entering its corresponding number from the list. For inserts, updates, and deletes, no output is printed. For retrieves and retrieve-commons, output is shown. If no output is shown, then the request had no qualifying records. Second, redisplay the traffic units in the list. Third, input a new traffic unit which will automatically be executed. The creation of the new traffic unit follows the same procedures as the "NEW LIST" option. The new traffic unit is only temporary and is not inserted into the list of requests. Fourth, upon completion of processing the traffic units, the user returns to the Request-Interface Menu.

The next option "NEW DATABASE" is chosen. This option is for choosing a new database to work with. If more than one database has been loaded, according to the procedures outlined in "Loading A Database" section, then this option allows the user to switch between different databases defined in the system. The user is only prompted for the database id. Note, this new database may require entering a new traffic unit file.

The next option "REDIRECT OUTPUT" is chosen. This option is for specifying the output mode of the session. The menu displayed here is straight forward, allowing the user to direct the output to the terminal, a file, both terminal and file, or suppress the output.

The next option "PERFORMANCE TESTING" is chosen. This option is for enabling/disabling the internal and external performance measurement settings. This option is very involved and is not operational in this version of the manual.

The next option "MODIFY" is chosen. This option is for modifying a list of ABDL requests that have been stored in a file. This option presents a collection of menus for altering the requests in a file. We begin by being prompted to use the current traffic unit file, identified by the addition of a version number, "#1". We choose not to modify the current version but by entering the same file name, without the version number, we create a new version, "#2", which will be an updated version of the old file. Then each traffic unit will be displayed to the user individually. The user may add, modify, remove, or make no change to this traffic unit. We choose only to modify the first retrieve request to become a retrieve-common request. The user is now prompted as in the "NEW LIST" option. The RETRIEVE-COMMON request is chosen. The source retrieve and target retrieve requests are built identical to the previously outlined retrieve requests. However,

after the source retrieve request is built, the user will be prompted for the common attributes. These are the attributes on whose values will do the merging operation. The domain types must be the same, i.e., either strings or integers. When all the traffic units from the file have been processed, additional requests may be added.

The final option "OLD LIST" is chosen. This option is for executing all ABDL requests in a given file. This option is also very straight forward. The user is prompted to use the current traffic unit file. If not, enter the desired file name. The appropriate output will be displayed.

## B. RELATIONAL/SQL INTERFACE

The relational/SQL interface is menu driven, accepts input in the relational language syntax and translates it into the attribute-based data language for storage and retrieval. We begin with a description of the four primary database operations of SQL used in this thesis, then the creation and loading of new databases, and finally the generation and execution of queries.

### 1. SQL Description

In this section we describe only the basics of the four primary database operations of SQL; INSERT, DELETE, UPDATE, and SELECT. The many variations of these operations will not be discussed here.

The INSERT query inserts a new tuple into an existing relation. After identifying the relation and attributes, the values are listed in corresponding order to the attributes. For example:

```
insert into cors (cnum, plac, room):
 <'c123', 'span', 332>
```

will insert a tuple into the course relation, which includes attribute-value pairs of course number c123, place spanegal hall, and room number 332.

The DELETE query can remove either selected tuples or all tuples in a relation. Omission of the WHERE clause results in deletion of all tuples. For example:

```
delete cors
where cnum = 'c123'
```

will delete the tuple from the course relation, where course number is equal to c123.

The UPDATE query modifies tuples in a relation by performing two functions: selecting the appropriate tuple and changing the specified value. For example:

```
update stud
set gpa = 4
where cnum = 'c205'
```

will modify the tuple in the student relation, where course number is equal to c205, by changing the gpa to 4.

The SELECT query has many variations and retrieves the values of the attributes specified. The relation from which values are to be selected is also specified. The following example demonstrates only the required keywords of the select query:

```
select cnum, plac, room
from cors
```

and will retrieve from the course relation, all the values of course number, place, and room number.

2. Creating and Loading Meta Data

Here we discuss how to load a new database in the relational format. From the System Menu, the "relational/SQL interface" option is chosen. The Load/Process Database Menu is then displayed and the "load new database" option is chosen. The user will be queried for a database name and then the method desired to input the database, either from a file or the terminal. We will explain both methods. It should be noted, that if terminal entry mode is chosen the database/create file will be lost if a new one is loaded. Also, when exiting the MBDS/MLDS system, all information typed in will be lost. Therefore, it is advised that all create files be generated utilizing the UNIX VI Editor.

The terminal input mode is first chosen for the create file. This file is the equivalent of the ABDL template file. The user is instructed to enter each relation/transaction separately. The user lists the relations name, then each attribute, its type and size. When the create file is complete, the user will begin creating a file equivalent to the ABDL descriptor file. Each attribute of a relation will be presented, beginning with the first relation. The user will be prompted as to whether to include that attribute as an indexing attribute, and if so, what its type (A, B, or C) and value will be.

Alternately, if the "read in a group of creates from a file" option is chosen, the user will be prompted for the UNIX file name, then queried about choosing indexing attributes.

After loading a new database by either method, control returns to the Load/Process Database Menu.

3. Generating and Executing Queries

Now we discuss the processing of a database previously loaded. From the Load/Process Database Menu, the "process existing database" option is chosen. The user will be queried for a database name, which must be the same as the previously loaded database, and then the method desired to input queries, either from a file or the terminal. In addition, the user may display the current database schema. Both methods of input will be explained. It should be noted, that if the terminal entry mode is chosen for the query/request file, as with the create file, the file will be lost if it is modified/updated or a new file is loaded. Also, when exiting the MBDS/MLDS system, all information typed in will be lost. Therefore, it is advised that all files be created and modified/updated utilizing the UNIX VI Editor.

The first option chosen is "display the current database schema". The schema equates to the ABDL template file. The differences include: the addition of the attribute name type length and a key value. The key value is set during the create table process but is not currently being utilized. It will not be discussed here.

The "read in queries from the terminal" option is now chosen. The user is prompted to enter queries/transactions separately. The queries are input in the format described previously and can be placed in any desired order.

On the other hand, if the "read in a group of queries from a file" option is chosen, the user will be prompted for the UNIX file name.

Upon completion of the query inputs, by either method, the query file is redisplayed with each query numbered sequentially and the Execution Menu is displayed. It should be noted, that the size/length of all database inputs are equivalent to those

specified in the ABDL interface. The Execution Menu allows the user to either redisplay the file of queries or to execute a query by entering its corresponding number from the list. Under normal operation, for the insert, delete, and update queries, only a completion statement would appear. For select queries output is shown. If no output is shown, then the query had no qualifying tuples. However, a translation of all queries into the attribute-based language will be shown. Upon completion of executing the desired queries, the user may exit back through the various menus to either change operations, change languages or exit to the operating system.

## C. HIERARCHICAL/DL/1 INTERFACE

The hierarchical/DL/1 interface is menu driven and accepts input in the hierarchical language syntax and translates it into the attribute-based data language for storage and retrieval. The heirarchical interface is nearly identical in both menu selections and operations to the relational interface. After a description of the four primary database operations of DL/1 used in this thesis, we will identify the differences between both interfaces, pertaining to the creation and loading of a new database and the generation and execution of queries.

### 1. DL/1 Description

In this section we describe only the basics of the four primary database operations of DL/1; INSERT, DELETE, REPLACE, and GET. The many variations of these operations will not be discussed here.

The INSERT query inserts a new segment occurrence into an existing segment. After identifying the field names, the values are listed in corresponding order. Then the segment or segments with a field are listed for insertion to a parent or child, respectively,

for example:

```
build (csnum, title, descripn):
 ('C100', 'database', 'intro')
isrt course

build (date, location, format):
 ('d1', 'monterey', 'mw')
isrt course (csnum = 'c100')
    offering
```

The first query will insert a segment occurrence into a parent/root segment course. The second, will insert into a child segment offering with a parent/root segment where course number is equal to c100.

The DELETE query will remove selected segment occurrences from a segment following a get statement. We must remember though, that the deletion of an occurrence causes the deletion of all its descendant occurrences from the database. For example:

```
ghu course (csnum = 'c100')
     offering (date = 'd2')
dlet
```

we first get the parent/root course where course number is equal to c100, then get the child offering where date is equal to d2, and then delete the selected occurrence. In this case, there are no descendants of the deleted occurrence.

The REPLACE query modifies an occurrence following its retrieval by a get statement. The same get operation as the delete query. For example:

```
ghu course (csnum = 'c100')
     offering (date = 'd3')
change location to 'washdc'
repl
```

we first get the parent/root course where course number is equal to c100, then get the child offering where date is equal to d3, and then modify the location value to washdc.

The GET query has many variations. It can be used for non-sequential processing of the database, as demonstrated with the delete and replace queries. However, its main function is to set or reset the position pointer to a specific occurrence of a specific type for subsequent sequential processing. For example:

```
        gu course

    aa gn course
       goto aa
```

the get unique will retrieve the first occurrence from the root course. The get next, without resetting the position pointer (explained later), will loop in the root segment and retrieve the rest of the occurrences.

2. Creating and Loading Meta Data

In this section, we discuss how to load a new database in the hierarchical format. Since this section is nearly identical to the corresponding section in the relational interface, we will only discuss the differences.

First, when loading a new database, the only mode of input for the database description will be to read it from a file.

Second, when creating a file that is equivalent to the ABDL descriptor file, there is the addition of the option 'n'. This option eliminates the displaying if all the segments and field names, if indexes are not desired for the database.

3. Generating and Executing Queries

In this section, we discuss the processing of a database previously loaded. Again, since this section is nearly identical to the corresponding section in the relational interface, we will only discuss the differences.

First, when processing an existing database, the mode of input menu does not contain the option to display the current database schema.

Second, the Execution Menu contains an option unique to the hierarchical interface. When executing a request, the currency pointer must be reset to the root after each action unless the user desires to retrieve more than one segment occurrence from the same segment (looping).

## D. NETWORK/CODASYL INTERFACE

The network/CODASYL interface is menu driven and accepts input in the network language syntax and translates it into the attribute-based data language for storage and retrieval. The network interface is nearly identical in both menu selections and operations to the relational interface. After a description of the four primary database operations of CODASYL used in this thesis, we will identify the differences between both interfaces, pertaining to the creation and loading of a new database and the generation and execution of queries.

1. CODASYL Description

In this section we describe the basics of the four primary database operations of CODASYL; STORE, ERASE, MODIFY, and GET. The many variations of these operations will no be discussed here. In addition, there are two more essential operations. A FIND operation, is used in combination with the ERASE, MODIFY, and

GET statements. The FIND operation locates an existing record occurrence and establishes it as the current run-unit, so the other operations may complete their manipulation. A MOVE operation, is an assignment statement that places the specified value in the template for the appropriate attribute.

The STORE query creates a new record occurrence, establishes it as the current run-unit, and inserts it into the database. Here, the move statement makes the assignment of values to attributes, then the new record is inserted. For example:

```
MOVE SS1 TO SNO IN SA
MOVE DEC TO SNAME IN SA
MOVE MONT TO CITY IN SA
STORE SA
```

will insert a new record with the following attribute value pairs, sno-ss1, sname-dec, and city-mont in sa.

The ERASE query deletes the current run-unit from the database. The move and find statements identifies the record and then establishes it as the current run-unit, respectively. For example:

```
MOVE SS2 TO SNO IN SA
FIND ANY SA USING SNO IN SA
ERASE SA
```

will delete the record where sno equals ss2 in sa.

The MODIFY query updates the current run-unit. As in the delete query, the move and find statements identify the record to be manipulated. The complete record or selective attribute values may be updated. For example:

```
MOVE SS1 TO SNO IN SA
MOVE SONY TO SNAME IN SA
MOVE TOKYO TO CITY IN SA
```

```
FIND ANY SA USING SNO IN SA
MODIFY SNAME, CITY IN SA
```

will update only sname to sony and city to tokyo, where sno equals ss1 in sa.

The GET query retrieves the current run-unit. Again, the move and find statements must be utilized to identify the record to be retrieved. Also, the record may be retrieved in part or whole. For example:

```
MOVE SS1 TO SNO IN SA
FIND ANY SA USING SNO IN SA
GET SA
```

will retrieve the complete record, where sno equals ss1 in sa.

2. Creating and Loading Meta Data

In this section, we discuss how to load a new database in the network format. However, this section has exactly the same differences to the corresponding section in the relational interface, as the hierarchical interface noted. Therefore, we refer to the corresponding section in the hierarchical interface for the discussion of differences.

3. Generating and Executing Queries

In this section, we discuss the processing of a database previously loaded. Again, since this section is nearly identical to the corresponding section in the relational interface, we will only discuss the differences.

First, there are no completion statements following the store, erase, and modify queries. However, a translation of all queries, into the attribute-base language, is provided.

Second, this implementation of CODASYL does not support a looping construct. Therefore, each set of attribute values in a record type must be individually

retrieved (get operation). Due to this limitation, a get operation should not be performed on a previously erased set of attribute values because any reference to a non-existant set will cause a system failure.

# V. CONCLUSION

## A. SUMMARY

As discussed in the introduction, it is essential to provide the instructor and students with effective pedagogical aids and software for the purpose of instructing modern database system architecture. This thesis has accomplished this goal by identifying the necessary pedagogical issues and implementing a detailed and complete lab manual to utilize the MBDS/MLDS system for instruction. In addition, an on-line version of the MBDS/MLDS User's Manual has been provided to the student to view or print, as required.

## B. DIFFICULTIES ENCOUNTERED/LIMITATIONS

The user must remember, when using the terminal entry mode to create a new database or request file, the information is not saved to reusable files. Therefore, any modification to or new files loaded, from the UNIX operating system, will cause the permanent loss of typed-in data.

The user must also be aware of the input size limitations listed in the ABDL interface section of the manual, i.e., record size, attribute name and value, etc. These limitations are applicable to the other language interfaces, due to the translation of each into ABDL for storage. This unfortunately limits the size of databases that can be created and executed on the system. The listed and other internal constants of ABDL, serve as an aid in the evaluation of MBDS performance. A change in one constant may

necessitate the modification of other interrelated constants. For the moment, the users must remain within the limitations.

## C. RECOMMENDATIONS FOR FUTURE EFFORTS

First, correct the loss of data when utilizing the terminal entry mode, as explained above.

Second, since the relational, hierarchical, and network interfaces have only slight differences, an effort to ensure identical menu options and output, with the exception of those areas unique to a language, would be beneficial to the user for consistency and ease of learning. The following areas should be identical in all language interfaces (in parenthesis are the languages that have implemented the area): (1) an option to load a new database from the terminal or a file (relational only), (2) addition of the 'n' option, when queried about indexing attributes (hierarchical and network only), (3) an option to display the current database schema (relational and network only), and (4) completion statements for the three primary operations which do not provide output: insert, modify, delete (relational and hierarchical only).

# THE MBDS/MLDS USER'S MANUAL

Department of Computer Science
Naval Postgraduate School
Monterey, California 93943

This manual provides the user with an instructional guide to the multi-backend and multi-lingual database systems (MBDS and MLDS). There are three parts to the manual. In the first part, we provide the user with the basic system commands and procedures. This includes logging onto the computer, initiating a session with MBDS/MLDS, terminating a session, and recovering from system crashes.

In the second part of the manual we provide the user with a guide to accessing the attribute-based data model and its data language, ABDL, through the attribute-based/ABDL interface to MBDS/MLDS. The attribute-based/ABDL interface is the native mode of MBDS. Through this interface, the user can generate new databases, load new databases, execute requests against existing databases, and evaluate the system using timing mechanisms.

Finally, in the third part of the manual, we provide the user with a separate guide for executing each of the remaining language interfaces that constitute the MLDS portion of our system, i.e., the relational/SQL, the hierarchical/DL/I, the network/CODASYL-DML and the functional/Daplex interfaces. Using these guides, the user can load new databases and execute transactions against existing databases using any of the four different data model/data language combinations.

# 1. SYSTEM STARTUP AND RECOVERY PROCEDURES.

In this part of the manual we discuss how to use MBDS/MLDS from a system perspective. The first step when using MBDS/MLDS is to log onto the computer system where MBDS/MLDS resides. Consult the MBDS/MLDS system manager (Albert Wong, SP-525A, x3009) for instructions and accounts. After logging onto the appropriate computer system, there are four basic commands for MBDS/MLDS at the system level: manmbds, pmanmbds, runmbds, stopmbds. The manmbds command is used to view this manual on the terminal screen. The pmanmbds command is used to print a copy of this manual out on the line printer.

The runmbds command is used to initiate a session with MBDS/MLDS. After typing runmbds, there is about a sixty second delay before a session can begin. When the menu shown below appears the system is ready to go.

Welcome to the MBDS/MLDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION->

This menu is referred to as the system menu. Selecting the 'a' option enables the attribute-based/ABDL interface portion of MBDS/MLDS (see part 2 of this manual). Selecting either the 'r', 'h', 'n' or 'f' options enables the MLDS portion of MBDS/MLDS (see part 3 of this manual). Lastly, selecting the 'x' option provides a graceful exit from MBDS/MLDS.

Finally, the stopmbds command is used when a graceful exit from the system is not possible. In some instances, it is possible that MBDS/MLDS will stop working for a variety of reasons. If the terminal is locked, type a 'cntrl C' (you may need more than one), and the terminal prompt should appear (i.e., isiv7%). Once you have obtained the terminal prompt, simply type stopmbds. This command will halt all of the appropriate MBDS/MLDS processes. You can then restart the system. If you have any trouble or are unsure of what to do, please contact the MBDS/MLDS system manager (i.e., Albert Wong).

## 2. USING THE ATTRIBUTE-BASED/ABDL INTERFACE.

In this, the second part of the MBDS/MLDS manual, we explore the actions and capabilities of the attribute-based/ABDL interface to MBDS/MLDS. These actions are enabled when the 'a' option of the system menu is chosen (shown below). In the remainder of this and other sections, text that is enclosed in comments (i.e., /* ..... */) is used as a means to annotate different aspects of the menus and options being presented.

/* The System Menu */

Welcome to the MBDS/MLDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> a

When the 'a' option has been picked, the attribute-based menu, shown below, is displayed. The attribute-based menu has four options, 'g', 'l', 'r' and 'x'. We examine each option in turn.

/* The Attribute-Based Menu */

Welcome to the attribute-based/ABDL interface

(g) - generate a new database
(l) - load a new database
(r) - execute the request interface
(x) - exit to the previous system menu

OPTION-> g

### 2.1. THE GENERATE-DATABASE MENU

The 'g' option is used to generate new databases in attribute-based form. When the 'g' option is picked, the generate-database menu is displayed. In the next few pages, we take the reader through all of the possible options of the generate-database menu. These options include:

(1) t - a collection of menus for generating the record-template file, which contains the meta-data for the different record types in our database.

(2) d - a collection of menus for generating the descriptor file, which contains the directory attributes of the database, along with possible initial values for the descriptors of each directory attribute.

(3) m - a collection of menus for generating (and optionally modifying) data sets for each of the attributes in the database. These data sets are then used to systematically generate arbitrary records for the database using the 'r' option.

(4) r - a collection of menus for generating the record file, which contains a group of records that are to be mass loaded by MBDS. Together, the 'm' and 'r' options can be used to generate test or sample databases. These test or sample databases contain records that have been systematically constructed from the sets of values created by the 'm' option. Through these two options, the user can quickly set up a test or sample database to experiment with.

(5) x - return to the previous menu, i.e., the attribute-based menu.

We now proceed to demonstrate each of these options in turn. The database that we will be using for our examples is the school database, and contains two types of records, one type for courses and a second type for students.

PLEASE NOTE THE FOLLOWING INFORMATION WHEN GENERATING YOUR DATABASE:

(1) Maximum record length is 50 characters, which includes the total length of the attribute names and values.

(2) All attribute names must begin with a minimum of two letters followed by zero or more letters or integers.

(3) All attribute values must begin with a letter, for a character string, or an integer followed by zero or more letters or integers.

/* The Generate-Database Menu */

What operation would you like to perform?

(t) - generate record template
(d) - generate descriptors
(m) - generate/modify sets
(r) - generate records
(x) - quit, return to previous menu
to load, execute or exit system

OPTION-> t /* The generate record template option is chosen */

ENTER THE NAME OF THE FILE TO BE USED TO STORE

```
TEMPLATE INFORMATION: scht.f
 /* For schOOL tEMPLATE
    NOTE: Maximum UNIX file name length is 13 characters. */

ENTER DATABASE ID: SCHL
 /* NOTE: Maximum database id length is 8 characters. */

/* Remember, TEMPLATES are the same as FILES */

ENTER THE NUMBER OF TEMPLATES FOR DATABASE SCHL: 2
 /* NOTE: Maximum digit length is 2 characters. */

ENTER THE NUMBER OF ATTRIBUTES FOR TEMPLATE #1: 4

ENTER THE NAME OF TEMPLATE #1: Cors
 /* NOTE: Maximum ABDL file name length is 10 characters. */

/* The different record types that are defined for the database
   must all have the first attribute to serve as an identification
   attribute for the record type.  For example, FILE, TEMPLATE,
   RECTYPE are all examples of such an attribute.  All of the
   record types that are created must use the same attribute to
   identify the record types. */

ENTER ATTRIBUTE NAME #1 FOR TEMPLATE Cors: FILE
 /* NOTE: Maximum attribute name length is 10 characters. */

ENTER VALUE TYPE (s=string, i=integer): s

ENTER ATTRIBUTE NAME #2 FOR TEMPLATE Cors: CNUM

/* The value type specifies the domain of the attribute
   values for the attribute CNUM */

ENTER VALUE TYPE (s=string, i=integer): s

ENTER ATTRIBUTE NAME #3 FOR TEMPLATE Cors: PLAC

ENTER VALUE TYPE (s=string, i=integer): s

ENTER ATTRIBUTE NAME #4 FOR TEMPLATE Cors: ROOM

ENTER VALUE TYPE (s=string, i=integer): i

ENTER THE NUMBER OF ATTRIBUTES FOR TEMPLATE #2: 3
```

ENTER THE NAME OF TEMPLATE #2: Stud

ENTER ATTRIBUTE NAME #1 FOR TEMPLATE Stud: FILE
/* In the Cors template, we also used FILE */

ENTER VALUE TYPE (s=string, i=integer): s

ENTER ATTRIBUTE NAME #2 FOR TEMPLATE Stud: NAME

ENTER VALUE TYPE (s=string, i=integer): s

ENTER ATTRIBUTE NAME #3 FOR TEMPLATE Stud: GPA

ENTER VALUE TYPE (s=string, i=integer): i

/* At this stage, our two record types are:

(<FILE, Cors>, <CNUM, s>, <PLAC, s>, <ROOM, s>)
(<FILE, Stud>, <NAME, s>, <GPA, i>)

The s and i represent value types (string and integer).

The template file, scht.f is:

SCHL
2
4
Cors
FILE s
CNUM s
PLAC s
ROOM i
3
Stud
FILE s
NAME s
GPA i

and may be viewed using the ls and cat options available
in the unix system after exiting from MBDS/MLDS. */

/* After we are done generating the template file, control returns
to the generate-database menu. */

What operation would you like to perform?

(t) - generate record template
(d) - generate descriptors
(m) - generate/modify sets
(r) - generate records
(x) - quit, return to previous menu
to load, execute or exit system

OPTION-> d /* The generate descriptors option is chosen */

ENTER THE NAME OF TEMPLATE FILE: scht.f
/* You must have already created a template file
before you begin loading descriptors. */

ENTER THE NAME OF THE FILE TO BE USED FOR
STORING DESCRIPTORS: schd.f
/* d for dESCRIPTOR */

/* For each attribute in the database, the user is queried as to whether
it is a directory attribute, and if so, what the descriptor type and
descriptors will be. */

Do you want attribute 'FILE' to be a directory attribute? (y/n) > y
/* FILE is a directory attribute - must always be one!!!! */

ENTER THE DESCRIPTOR TYPE FOR FILE:(A,B,C) > b

/* Lower bound as ! indicates a type B or type C equality descriptor */

Use '!' to indicate that no lower bound exists ... Enter '@' to stop
Note: '@' Must be Entered When the Lower Bound is Requested

ENTER LOWER BOUND FOR DESCRIPTOR: !
/* NOTE: Maximum lower and upper bound lengths are 10 characters. */

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = !): Cors

ENTER LOWER BOUND FOR DESCRIPTOR: !

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = !): Stud

ENTER LOWER BOUND FOR DESCRIPTOR: @
/* End of list of descriptors for the FILE attribute */

Do you want attribute 'CNUM' to be a directory attribute? (y/n) > y

ENTER THE DESCRIPTOR TYPE FOR CNUM:(A,B,C) > a

```
/* When using range descriptors (type-A) you must be sure your
   ranges are mutually exclusive.  MBDS does not check this!!! */

Use  '!'  to indicate that no lower bound exists ... Enter '@' to stop
 Note: '@' Must be Entered When the Lower Bound is Requested

ENTER LOWER BOUND FOR DESCRIPTOR: C100

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = C100): C199

ENTER LOWER BOUND FOR DESCRIPTOR: C250

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = C250): C400

ENTER LOWER BOUND FOR DESCRIPTOR: @

Do you want attribute 'PLAC' to be a directory attribute? (y/n) > n
  /* Not a directory attribute */

Do you want attribute 'ROOM' to be a directory attribute? (y/n) > n

Do you want attribute 'NAME' to be a directory attribute? (y/n) > y

ENTER THE DESCRIPTOR TYPE FOR NAME:(A,B,C) > a

Use  '!'  to indicate that no lower bound exists ... Enter '@' to stop
 Note: '@' Must be Entered When the Lower Bound is Requested

ENTER LOWER BOUND FOR DESCRIPTOR: Aaa

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = Aaa): Lzz

ENTER LOWER BOUND FOR DESCRIPTOR: Maa

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = Maa): Rzz

ENTER LOWER BOUND FOR DESCRIPTOR: Saa

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = Saa): Zzz

ENTER LOWER BOUND FOR DESCRIPTOR: @

Do you want attribute 'GPA' to be a directory attribute? (y/n) > y

ENTER THE DESCRIPTOR TYPE FOR GPA:(A,B,C) > b
```

Use '!' to indicate that no lower bound exists ... Enter '@' to stop
Note: '@' Must be Entered When the Lower Bound is Requested

ENTER LOWER BOUND FOR DESCRIPTOR: !

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = !): 1

ENTER LOWER BOUND FOR DESCRIPTOR: !

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = !): 2

ENTER LOWER BOUND FOR DESCRIPTOR: !

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = !): 3

ENTER LOWER BOUND FOR DESCRIPTOR: !

ENTER UPPER BOUND FOR DESCRIPTOR ... (lower bound = !): 4

ENTER LOWER BOUND FOR DESCRIPTOR: @

/* At this stage, the descriptor file is:

```
SCHL
FILE b s
! Cors
! Stud
@
CNUM a s
C100 C199
C250 C400
@
NAME a s
Aaa Lzz
Maa Rzz
Saa Zzz
@
GPA b i
! 1
! 2
! 3
! 4
@
$       /* Automatically inserted by MBDS */
```

and may be viewed using the ls and cat options available

in the unix system after exiting from MBDS/MLDS. */

/* After we are done generating the descriptor file, control returns
to the generate-database menu. */

What operation would you like to perform?

(t) - generate record template
(d) - generate descriptors
(m) - generate/modify sets
(r) - generate records
(x) - quit, return to previous menu
to load, execute or exit system

OPTION-> m /* The generate/modify sets option is chosen */

ENTER THE NAME OF TEMPLATE FILE: scht.f

CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE 'FILE' ON TEMPLATE 'Cors':
(n) - generate a new set for it
(m) - modify an existing set for it
(s) - do nothing with it

OPTION-> s /* We are not going to generate a set of values for the FILE
attribute, since there are already two values for it, namely,
Cors and Stud, corresponding to the two different files. */

CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE 'CNUM' ON TEMPLATE 'Cors':
(n) - generate a new set for it
(m) - modify an existing set for it
(s) - do nothing with it

OPTION-> n /* Generate a set of values for the CNUM attribute */

ENTER THE NAME OF THE FILE TO BE USED TO STORE THE SET: cnum.s
/* Store the values in the file cnum.s */

ENTER SET VALUE: C123
/* NOTE: Maximum attribute value length is 10 characters. */

ENTER SET VALUE: C135

ENTER SET VALUE: C205

```
ENTER SET VALUE: @
 /* End of list of values for the CNUM attribute */

Set generation completed ... do you want to modify it? (y/n) > n
 /* Do not modify the list */

CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE 'PLAC' ON TEMPLATE 'Cors':
      (n) - generate a new set for it
      (m) - modify an existing set for it
      (s) - do nothing with it

OPTION-> n

ENTER THE NAME OF THE FILE TO BE USED TO STORE THE SET: plac.s

ENTER SET VALUE: Span

ENTER SET VALUE: Root

ENTER SET VALUE: @

Set generation completed ... do you want to modify it? (y/n) > n

CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE 'ROOM' ON TEMPLATE 'Cors':
      (n) - generate a new set for it
      (m) - modify an existing set for it
      (s) - do nothing with it

OPTION-> n

ENTER THE NAME OF THE FILE TO BE USED TO STORE THE SET: room.s

ENTER SET VALUE: 117

ENTER SET VALUE: 205

ENTER SET VALUE: 332

ENTER SET VALUE: @

Set generation completed ... do you want to modify it? (y/n) > n

CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE 'FILE' ON TEMPLATE 'Stud':
```

(n) - generate a new set for it
(m) - modify an existing set for it
(s) - do nothing with it

OPTION-> s

CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE 'NAME' ON TEMPLATE 'Stud':
(n) - generate a new set for it
(m) - modify an existing set for it
(s) - do nothing with it

OPTION-> n

ENTER THE NAME OF THE FILE TO BE USED TO STORE THE SET: name.s

ENTER SET VALUE: Steve

ENTER SET VALUE: Jud

ENTER SET VALUE: Greg

ENTER SET VALUE: Jean

ENTER SET VALUE: @

Set generation completed ... do you want to modify it? (y/n) > n

CHOOSE ACTION TO BE TAKEN FOR
ATTRIBUTE 'GPA' ON TEMPLATE 'Stud':
(n) - generate a new set for it
(m) - modify an existing set for it
(s) - do nothing with it

OPTION-> n

ENTER THE NAME OF THE FILE TO BE USED TO STORE THE SET: gpa.s

ENTER SET VALUE: 1

ENTER SET VALUE: 2

ENTER SET VALUE: 3

ENTER SET VALUE: 4

ENTER SET VALUE: @

Set generation completed ... do you want to modify it? (y/n) > n

/* While we have not generated the 'm' option, its use allows
the user to add or subtract values from the set of values
that have already been specified for a particular attribute */

/* At this point, we have generated five files of values, namely,
cnum.s, plac.s, room.s, name.s and gpa.s. These files are shown
below:

| cnum.s | plac.s | room.s | name.s | gpa.s |
|---|---|---|---|---|
| C123 | Span | 117 | Steve | 1 |
| C135 | Root | 205 | Jud | 2 |
| C205 | $ | 332 | Greg | 3 |
| $ | | $ | Jean | 4 |
| | | | $ | $ |

and may be viewed using the ls and cat options available
in the unix system after exiting from MBDS/MLDS. */

/* After we are done generating/modifying the value sets, control returns
to the generate-database menu. */

What operation would you like to perform?

(t) - generate record template
(d) - generate descriptors
(m) - generate/modify sets
(r) - generate records
(x) - quit, return to previous menu
to load, execute or exit system

OPTION-> r /* The generate records option is chosen */
/* NOTE: Maximum records produced is 10,000. */

ENTER THE NAME OF TEMPLATE FILE: scht.f

ENTER THE NAME OF THE FILE TO BE USED FOR STORING RECORDS: schr.f
/* r for rECORDS */

ENTER THE NAME OF THE FILE CONTAINING THE
VALUES FOR ATTRIBUTE 'CNUM' ON TEMPLATE 'Cors': cnum.s

ENTER THE NAME OF THE FILE CONTAINING THE

VALUES FOR ATTRIBUTE 'PLAC' ON TEMPLATE 'Cors': plac.s

ENTER THE NAME OF THE FILE CONTAINING THE
VALUES FOR ATTRIBUTE 'ROOM' ON TEMPLATE 'Cors': room.s

/* 18 is the product of 3, 2 and 3, the number of values in the
cnum.s, plac.s and room.s files, respectively. */

18 records can be generated for template 'Cors'...

How many records do you want generated? 6
/* Generate 6 of these for mass loading */

ENTER THE NAME OF THE FILE CONTAINING THE
VALUES FOR ATTRIBUTE 'NAME' ON TEMPLATE 'Stud': name.s

ENTER THE NAME OF THE FILE CONTAINING THE
VALUES FOR ATTRIBUTE 'GPA' ON TEMPLATE 'Stud': gpa.s

16 records can be generated for template 'Stud'...

How many records do you want generated? 4

**********

ALL RECORDS GENERATED

**********

/* At this stage, the record file may look like this:

SCHL
@
Cors
C123 Root 332
C123 Span 332
C205 Root 117
C205 Root 332
C123 Root 205
C123 Span 205
@
Stud
Steve 4
Jean 1

Jud 2
Steve 1
$

and may be viewed using the ls and cat options available in the
unix system after exiting from MBDS/MLDS.
Note: records are generated randomly and will produce a different
set each time the record file is created. */

/* After we are done generating records, control returns
to the generate-database menu. */

What operation would you like to perform?

(t) - generate record template
(d) - generate descriptors
(m) - generate/modify sets
(r) - generate records
(x) - quit, return to previous menu
to load, execute or exit system

OPTION-> x /* The quit option is chosen, return to
the attribute-based menu */

This completes our presentation of the generate-database menu and its options. We now proceed to the next option of the attribute-based menu, namely, the 'l' option.

## 2.2. THE LOAD-DATABASE MENU

The 'l' option is used to load the newly generated database in attribute-based form. When the 'l' option is picked, the load-database menu is displayed. In the next few pages, we take the reader through all of the possible options of the load-database menu. These options include:

(1) t - used to load the template and descriptor files for a new database.

(2) r - used to mass-load the record file for a new (or existing) database.

(3) x - return to the previous menu, i.e., the attribute-based menu.

We now proceed to demonstrate each of these options in turn. We continue to use the school database in our examples.

/* The Attribute-Based Menu */

Welcome to the attribute-based/ABDL interface

(g) - generate a new database
(l) - load a new database
(r) - execute the request interface
(x) - exit to the previous system menu

OPTION-> l /* The load a new database option is chosen */

/* The Load-Database Menu */

What operation would you like to perform?

(t) - load the template and descriptor files
(r) - mass load a file of records
(x) - exit, return to previous menu

OPTION-> t /* The load template and descriptor files option is chosen */

ENTER NAME OF FILE CONTAINING TEMPLATE INFORMATION: scht.f

/* Note: In some versions of MBDS/MLDS, there may be a number of
diagnostic messages that are printed at this point. Please
ignore them. */

ENTER NAME OF FILE CONTAINING THE DESCRIPTORS: schd.f

/* Note: In some versions of MBDS/MLDS, there may be a number of
diagnostic messages that are printed at this point. Please
ignore them. */

/* Return to the load-database menu */

What operation would you like to perform?

(t) - load the template and descriptor files
(r) - mass load a file of records
(x) - exit, return to previous menu

OPTION-> r /* The mass load of the record file option is chosen */

/* Please read the important note!!!! */

NOTE TO THE USER!!!! YOU MUST HAVE LOADED THE TEMPLATES
AND DESCRIPTORS FOR A DATABASE, BEFORE ATTEMPTING TO LOAD
ANY RECORDS INTO THE DATABASE!!!!

Do you wish to continue? (y/n) > y

ENTER NAME OF FILE CONTAINING RECORDS TO BE LOADED: schr.f

/* Note: In some versions of MBDS/MLDS, there may be a number of
diagnostic messages that are printed at this point. Please
ignore them.
If your record file has been incorrectly assembled due to
improper data in the template or descriptor files, MBDS/MLDS
may stop working. If this happens, follow the RECOVER
PROCEDURE at the beginning of this manual. */

/* At this stage, a "Record count" of all records in the record
file will be listed. */

/* Return to the load-database menu */

What operation would you like to perform?

(t) - load the template and descriptor files
(r) - mass load a file of records
(x) - exit, return to previous menu

OPTION-> x /* The exit option is chosen */

At this point, we are done loading a database, so control returns to the attribute-based menu. Now, we proceed to the 'r' option, to execute the request interface.

## 2.3. THE REQUEST-INTERFACE MENU

The 'r' option is used to execute the request interface for attribute-based databases and to process ABDL requests and transactions. When the 'r' option is picked, the request-interface menu is displayed. In the next few pages, we take the reader through all of the possible options of the request-interface menu. These options include:

(1) s - an option for selecting a file of previously created ABDL requests. This option presents a menu for displaying and submitting these requests for processing.

(2) n - an option for creating a new file of ABDL requests. The collection of menus for this option allows the user to create a file of INSERT, DELETE, UPDATE, RETRIEVE and RETRIEVE-COMMON requests. By using the menus, correct syntax is guaranteed.

(3) d - an option for choosing a new database to work with. This option allows the user to switch between different databases defined in the system.

(4) r - an option for specifying the output mode of the session. This option allows the user to direct the output to the terminal, a file, or suppress output.

(5) p - an option for enabling/disabling the internal and external performance measurement hooks. This option is very involved and is not presented in this version of the manual. It will be presented in a later version of this manual.

(6) m - an option for modifying a list of ABDL requests that have been stored in a file. This option presents a collection of menus for altering the requests in the file.

(7) o - an option for executing all ABDL requests in a given file.

(8) x - return to the previous menu, i.e., the attribute-based menu.

We now proceed to demonstrate each of these options in turn. We continue to use the school database in our examples.

/* The Attribute-Based Menu */

Welcome to the attribute-based/ABDL interface

(g) - generate a new database
(l) - load a new database
(r) - execute the request interface
(x) - exit to the previous system menu

OPTION-> r /* The execute request interface option is chosen */

/* The Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an
existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MBDS/MLDS user's manual before choosing
the subsession options that are marked with an asterisk (*)

OPTION-> n /* The new list of traffic units option is chosen,
Traffic Unit is MBDS terminology for a request */

Enter the name for the traffic unit file
It may be up to 13 characters long including the .ext.
Filenames may include only one '#' character

as the first character before the version number.

FILE NAME-> schreq.f /* req for reqUESTS */

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> i /* First, an insert request */

/* For insert requests, it is important to note that data values
must be supplied for each attribute in the record type (file).
There is no such concept as a null value. */

INSERT Request

Begin entering keywords as you are prompted.
You will be prompted first for the 'Attribute' and then for the 'value'.
End each attribute or value with a single <return>.

When you have finished entering keywords, respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

VALUE-> Cors

ATTRIBUTE (<cr> to finish)-> CNUM

VALUE-> C333

ATTRIBUTE (<cr> to finsh)-> PLAC

VALUE-> Root

ATTRIBUTE (<cr> to finish)-> ROOM

VALUE-> 122

ATTRIBUTE (<cr> to finish)-> /* a return to finish the insert */

/* The insert request is:

[INSERT(<FILE,Cors>,<CNUM,C333>,<PLAC,Root>,<ROOM,122>)]

*/

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> r /* Second, a retrieve request */

RETRIEVE Request

/* The BY clause is used for sorting and is operational.
The WITH clause accesses records by their addresses and
is not currently scheduled for implementation. */

Enter responses as you are prompted. You will be prompted first for the predicates of the query, then attributes for the target-list, next for an attribute for the optional BY clause and finally for a pointer for the optional WITH clause.

When you have finished entering predicates for the query, respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE /* Always must use a file */

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a /* Type of operator */

VALUE-> Stud

So far your conjunction is
(FILE=Stud).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > y

ATTRIBUTE (<cr> to finish)-> NAME

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> d

VALUE-> John

So far your conjunction is
(FILE=Stud)and(NAME>=John).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

/* Append more conjunctions means to add a disjunction */

Do you wish to append more conjunctions to the query? (y/n) > n
/* If y, you must access the same file, in this case, Stud
There is no cross-file access in a retrieve */

Begin entering attributes for the Target-List. When you are
through entering attributes respond to the ATTRIBUTE> prompt with <return>.
Do you wish to be prompted for aggregation? (y/n) > n

ATTRIBUTE (<cr> to finish)-> NAME

ATTRIBUTE (<cr> to finish)-> GPA

ATTRIBUTE (<cr> to finish)-> /* A return ends target list */

Do you wish to use a BY clause? (y/n) > n
/* This ends the retrieve request */

/* The retrieve request is:

[RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)]

*/

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> r /* Third, a retrieve request with an aggregate operation */

RETRIEVE Request

Enter responses as you are prompted. You will be prompted first for the predicates of the query, then attributes for the target-list, next for an attribute for the optional BY clause and finally for a pointer for the optional WITH clause.

When you have finished entering predicates for the query, respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Stud

So far your conjunction is
(FILE=Stud).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > y

ATTRIBUTE (<cr> to finish)-> NAME

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> d

VALUE-> John

So far your conjunction is
(FILE=Stud)and(NAME>=John).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

Do you wish to append more conjunctions to the query? (y/n) > n

Begin entering attributes for the Target-List. When you are
through entering attributes respond to the ATTRIBUTE> prompt with <return>.
Do you wish to be prompted for aggregation? (y/n) > y
/* NOTE: Maximum aggregate operators per request is 5. */

ATTRIBUTE (<cr> to finish)-> GPA

Do you want to have an aggregate operator perform on this attribute? (y/n) > y

Enter the character for the desired Aggregate Operator

(a) AVG
(s) SUM
(c) COUNT
(x) MAX
(n) MIN

OPTION-> x

ATTRIBUTE (<cr> to finish)-> /* A return ends target list */

Do you wish to use a BY clause? (y/n) > n
/* This ends the retrieve request */

/* The retrieve request is:

[RETRIEVE((FILE=Stud)and(NAME>=John))(MAX(GPA))]

*/

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> r /* Fourth, a retrieve request with a BY clause */

RETRIEVE Request

Enter responses as you are prompted. You will be prompted first for the predicates of the query, then attributes for the target-list, next for an attribute for the optional BY clause and finally for a pointer for the optional WITH clause.

When you have finished entering predicates for the query, respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Stud

So far your conjunction is
(FILE=Stud).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > y

ATTRIBUTE(<cr> to finish)-> NAME

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> d

VALUE-> John

So far your conjunction is
(FILE=Stud)and(NAME>=John).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

Do you wish to append more conjunctions to the query? (y/n) > n

Begin entering attributes for the Target-List. When you are
through entering attributes respond to the ATTRIBUTE> prompt with <return>.
Do you wish to be prompted for aggregation? (y/n) > n

ATTRIBUTE (<cr> to finish)-> NAME

ATTRIBUTE (<cr> to finish)-> GPA

ATTRIBUTE (<cr> to finish)-> /* A return ends target list */

Do you wish to use a BY clause? (y/n) > y

ATTRIBUTE (<cr> to finish)-> GPA /* This ends the retrieve request */

/* The retrieve request is:

[RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)BY GPA]

*/

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> u /* Fifth, an update request */

UPDATE Request

/* The query of the update is the disjunctive normal form
collection of predicates, as with the retrieve */

Enter responses as you are prompted. You will be first
asked for the predicates necessary to build the query and then
the attribute and expression required to construct the modifier.

/* Query constructed as with the Retrieve request */

When you are finished entering predicates for the query,
respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Cors

So far your conjunction is
(FILE=Cors).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > y

ATTRIBUTE (<cr> to finish)-> CNUM

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> C205

So far your conjunction is
(FILE=Cors)and(CNUM=C205).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

Do you wish to append more conjunctions to the query? (y/n) > n

/* Now we enter the attribute whose values will be altered
during the update operation. The update request may only
modify a single attribute at a time. If more than one
attribute of a record must be modified, then multiple
update requests must be used. */

Enter the attribute-being-modified.

ATTRIBUTE (<cr> to finish)-> PLAC

Enter the number indicating the desired modifier type

(0) Set attribute equal to a constant
(1) Set attribute equal to a function of itself
(2) Set attribute equal to a function of another attribute
(3) Set attribute equal to a function of another attribute of the query
(4) Set attribute equal to a function of another attribute of a pointer

OPTION-> 0 /* Use only type-0 modifier. Types 1-4 are not implemented at this time */

Enter constant-> Span

/* The update request is:

[UPDATE((FILE=Cors)and(CNUM=C205))<PLAC=Span>]

*/

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> d /* Sixth, a delete request */

/* Just like a retrieve, only no target list */

DELETE Request

Enter responses as you are prompted. You will be asked to enter attributes, values, and relational operators as predicates for the query.

When you are finished entering predicates respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Stud

So far your conjunction is
(FILE=Stud).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > y

ATTRIBUTE (<cr> to finish)-> NAME

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Greg

So far your conjunction is
(FILE=Stud)and(NAME=Greg).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

Do you wish to append more conjunctions to the query? (y/n) > n

/* The delete request is:

[DELETE((FILE=Stud)and(NAME=Greg))]

*/

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> f /* Done with this task */

/* Return to the Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an
existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing
the subsession options that are marked with an asterisk (*)

OPTION-> s /* The select a list of traffic units option is chosen */

Will you use the current traffic unit file (schreq.f#1)? (y/n) > y
/* We use the file already created, i.e., the last file
which we accessed. If no, we input a file name. */

List of executable traffic units

(0) [INSERT(<FILE,Cors>,<CNUM,C333>,<PLAC,Root>,<ROOM,122>)]

(1) [RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)]

(2) [RETRIEVE((FILE=Stud)and(NAME>=John))(MAX(GPA))]

(3) [RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)BY GPA]

(4) [UPDATE((FILE=Cors)and(CNUM=C205))<PLAC=Span>]

(5) [DELETE((FILE=Stud)and(NAME=Greg))]

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> 0 /* A number executes the traffic unit
For inserts, updates and deletes,
no output is printed. For retrieves
and retrieve-commons, output is shown.
If no output is shown, then the request
had no qualifying records. */

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> 1 /* Execute the retrieve. */

(<NAME, Jud>, <GPA,2>)
(<NAME, Steve>, <GPA, 1>)
(<NAME, Steve>, <GPA, 4>)

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> 2 /* Execute the retrieve with an aggregate operation */

(<MAX(GPA),4>)

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> 3 /* Execute the retrieve with a BY clause */

(<NAME, Steve>, <GPA, 1>)
(<NAME, Jud>, <GPA, 2>)
(<NAME, Steve>, <GPA, 4>)

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> d /* A d displays the list again */

List of executable traffic units

(0) [INSERT(<FILE,Cors>,<CNUM,C333>,<PLAC,Root>,<ROOM,122>)]

(1) [RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)]

(2) [RETRIEVE((FILE=Stud)and(NAME>=John))(MAX(GPA))]

(3) [RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)BY GPA]

(4) [UPDATE((FILE=Cors)and(CNUM=C205))<PLAC=Span>]

(5) [DELETE((FILE=Stud)and(NAME=Greg))]

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> 4 /* Execute the update */

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> n /* Input a new traffic unit */

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> r

RETRIEVE Request

Enter responses as you are prompted. You will be prompted first for the predicates of the query, then attributes for the target-list, next for an attribute for the optional BY clause and finally for a pointerfor the optional WITH clause.

When you have finished entering predicates for the query, respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL

(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Stud

So far your conjunction is
(FILE=Stud).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

Do you wish to append more conjunctions to the query? (y/n) > n

Begin entering attributes for the Target-List. When you are
through entering attributes respond to the ATTRIBUTE> prompt with <return>.
Do you wish to be prompted for aggregation? (y/n) > n

ATTRIBUTE (<cr> to finish)-> NAME

ATTRIBUTE (<cr> to finish)-> GPA

ATTRIBUTE (<cr> to finish)->

Do you wish to use a BY clause? (y/n) > n

/* At this stage, the retrieve that has been created is:

[RETRIEVE ((FILE = Stud)) (NAME, GPA)]

*/

/* These are the results */

(<NAME, Steve>, <GPA, 1>)
(<NAME, Jud>, <GPA, 2>)
(<NAME, Jean>, <GPA, 1>)
(<NAME, Steve>, <GPA, 4>)

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> d /* Note that the new retrieve is not inserted
into our list of requests. */

List of executable traffic units

(0) [INSERT(<FILE,Cors>,<CNUM,C333>,<PLAC,Root>,<ROOM,122>)]

(1) [RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)]

(2) [RETRIEVE((FILE=Stud)and(NAME>=John))(MAX(GPA))]

(3) [RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)BY GPA]

(4) [UPDATE((FILE=Cors)and(CNUM=C205))<PLAC=Span>]

(5) [DELETE((FILE=Stud)and(NAME=Greg))]

Select Options

(d) redisplay the traffic units in the list
(n) enter a new traffic unit to be executed
(num) execute the traffic unit at [num]
from the above list
(x) exit from this SELECT subsession

OPTION-> x

/* Return to the Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an
existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing
the subsession options that are marked with an asterisk (*)

OPTION-> x /* To demonstrate the new database option ('d'), we exit
to the Attribute-Based Menu in order to load another

new database. */

Welcome to the attribute-based/ABDL interface

(g) - generate a new database
(l) - load a new database
(r) - execute the request interface
(x) - exit to the previous system menu

OPTION-> l

/* The database we are loading is named TEST and contains three
record (file) types, Part, Sups and Ship */

What operation would you like to perform?

(t) - load the template and descriptor files
(r) - mass load a file of records
(x) - exit, return to previous menu

OPTION-> t

ENTER NAME OF FILE CONTAINING TEMPLATE INFORMATION: tt.f

ENTER NAME OF FILE CONTAINING THE DESCRIPTORS: td.f

What operation would you like to perform?

(t) - load the template and descriptor files
(r) - mass load a file of records
(x) - exit, return to previous menu

OPTION-> r

NOTE TO THE USER!!!! YOU MUST HAVE LOADED THE TEMPLATES
AND DESCRIPTORS FOR A DATABASE, BEFORE ATTEMPTING TO LOAD
ANY RECORDS INTO THE DATABASE!!!

Do you wish to continue? (y/n) > y

ENTER NAME OF FILE CONTAINING RECORDS TO BE LOADED: tr.f

What operation would you like to perform?

(t) - load the template and descriptor files
(r) - mass load a file of records

(x) - exit, return to previous menu

OPTION-> x

/* At this stage, the current database for this user is the newly loaded database, namely the TEST database. We now proceed to the request-interface menu to demonstrate the 'd' option. */

/* The Attribute-Based Menu */

Welcome to the attribute-based/ABDL interface

(g) - generate a new database
(l) - load a new database
(r) - execute the request interface
(x) - exit to the previous system menu

OPTION-> r

/* The Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an
existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing the subsession options that are marked with an asterisk (*)

OPTION-> d /* The new database option is chosen */

/* The system asks us for the database id (name) to switch to */

Enter the database id

DATABASE ID > SCHL /* We return to the school database */

/* Return to the Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an
existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing
the subsession options that are marked with an asterisk (*)

OPTION-> r /* The redirect output option is chosen */

Enter the appropriate number for the outputform.
(1) Send output to CRT only
(2) Send output to File only
(3) Send output to both CRT and File
(4) Do not display output

OPTION-> 1 /* Send output to the CRT only */

/* Return to the Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an
existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing
the subsession options that are marked with an asterisk (*)

OPTION-> p /* The performance evaluation option is chosen */

/* This option is not operational in this version of MBDS/MLDS.
Later versions of MBDS/MLDS will have this option working and

will update this manual to explain this option. */

What would you like to do:

(e) Turn on external timer.
(i) Turn on internal timers.
(a) ABORT..Abandon all requested actions.
(x) Exit to previous menu.

OPTION-> x

/* Return to the Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list
(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an
existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing
the subsession options that are marked with an asterisk (*)

OPTION-> m /* The modify a list of traffic units option is chosen */

/* Versions of files are created and maintained by MBDS
The user may manipulate the files (and versions) using
the Unix operating system and its file manipulation commands. */

Will you use the current traffic unit file (schreq.f#1)? (y/n) > n

Enter the name for the traffic unit file
It may be up to 13 characters long including the .ext.
Filenames may include only one '#' character
as the first character before the version number.

FILE NAME-> schreq.f /* We modify the file we created earlier */

Do you want to give this modified file a new name?
Otherwise, it will be called the updated version of the oldfile. (y/n) > n
/* We use the same name, a new version is created */

[INSERT(<FILE,Cors>,<CNUM,C333>,<PLAC,Root>,<ROOM,122>)]

Enter the preferred change type for this traffic unit.

(a) ADD
(m) MODIFY
(r) REMOVE
(n) NO CHANGE

OPTION-> n /* We do not change the insert */

[RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)]

Enter the preferred change type for this traffic unit.

(a) ADD
(m) MODIFY
(r) REMOVE
(n) NO CHANGE

OPTION-> m /* We change the retrieve */

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> r

Enter the character for the desired next step.

(i) INSERT
(r) RETRIEVE
(u) UPDATE
(d) DELETE
(c) RETRIEVE COMMON

OPTION-> c /* A retrieve common is entered */

RETRIEVE COMMON Request

First enter the source retrieve request

RETRIEVE Request

Enter responses as you are prompted. You will be prompted first for the predicates of the query, then attributes for the target-list, next for an attribute for the optional BY clause and finally for a pointerfor the optional WITH clause.

When you have finished entering predicates for the query, respond to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Cors

So far your conjunction is
(FILE=Cors).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > y

ATTRIBUTE (<cr> to finish)-> CNUM

Enter the character for the desired relational operator

(a) = EQUAL
(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> d

VALUE-> C200

So far your conjunction is
(FILE=Cors)and(CNUM>=C200).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

Do you wish to append more conjunctions to the query? (y/n) > n

Begin entering attributes for the Target-List. When you are
through entering attributes respond to the ATTRIBUTE> prompt with <return>.
Do you wish to be prompted for aggregation? (y/n) > n

ATTRIBUTE (<cr> to finish)-> CNUM

ATTRIBUTE (<cr> to finish)-> PLAC

ATTRIBUTE (<cr> to finish)-> ROOM

ATTRIBUTE (<cr> to finish)->

Do you wish to use a BY clause? (y/n) > n

/* The Common Attributes are the attribute on whose values we will do
the merging operation. The domain types must be the same, i.e.,
either strings or integers. */

COMMON ATTRIBUTE 1> PLAC /* From the Cors file */

COMMON ATTRIBUTE 2> NAME /* From the Stud file */

The request being built is: /* The request so far */

[RETRIEVE((FILE=Cors)and(CNUM>=C200))(CNUM,PLAC,ROOM)
COMMON(PLAC,NAME)

Enter the target retrieve

RETRIEVE Request

Enter responses as you are prompted. You will be prompted first for
the predicates of the query, then attributes for the target-list,
next for an attribute for the optional BY clause and finally for
a pointer for the optional WITH clause.

When you have finished entering predicates for the query, respond
to the ATTRIBUTE> prompt with a <return>.

ATTRIBUTE (<cr> to finish)-> FILE

Enter the character for the desired relational operator

(a) = EQUAL

(b) /= NOT EQUAL
(c) > GREATER THAN
(d) >= GREATER THAN or EQUAL
(e) < LESS THAN
(f) <= LESS THAN or EQUAL

OPTION-> a

VALUE-> Stud

So far your conjunction is
(FILE=Stud).
Do you wish to 'and' additional predicates to this conjunction? (y/n) > n

Do you wish to append more conjunctions to the query? (y/n) > n

Begin entering attributes for the Target-List. When you are
through entering attributes respond to the ATTRIBUTE> prompt with <return>.
Do you wish to be prompted for aggregation? (y/n) > n

ATTRIBUTE (<cr> to finish)-> GPA

ATTRIBUTE (<cr> to finish)->

Do you wish to use a BY clause? (y/n) > n

/* The retrieve-common that replaces the retrieve */

The request being processed is:

[RETRIEVE((FILE=Cors)and(CNUM>=C200))(CNUM,PLAC,ROOM)
COMMON(PLAC,NAME)
RETRIEVE(FILE=Stud)(GPA)]

[RETRIEVE((FILE=Stud)and(NAME>=John))(MAX(GPA))]

Enter the preferred change type for this traffic unit.

(a) ADD
(m) MODIFY
(r) REMOVE
(n) NO CHANGE

OPTION-> n /* No change to the retrieve */

[RETRIEVE((FILE=Stud)and(NAME>=John))(NAME,GPA)BY GPA]

Enter the preferred change type for this traffic unit.

(a) ADD
(m) MODIFY
(r) REMOVE
(n) NO CHANGE

OPTION-> n /* No change to the retrieve */

[UPDATE((FILE=Cors)and(CNUM=C205))<PLAC=Span>]

Enter the preferred change type for this traffic unit.

(a) ADD
(m) MODIFY
(r) REMOVE
(n) NO CHANGE

OPTION-> n /* No change to the update */

[DELETE((FILE=Stud)and(NAME=Greg))]

Enter the preferred change type for this traffic unit.

(a) ADD
(m) MODIFY
(r) REMOVE
(n) NO CHANGE

OPTION-> n /* No change to the delete */

All the traffic units from file schreq.f#2 have been processed
Additional requests may now be added

Enter the character for the desired Traffic Unit type.

(r) Request
(t) Transaction (multiple requests)
(f) Finished entering traffic units.

OPTION-> f /* We now quit */

/* Return to the Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list

(or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units
(o) * OLD LIST; execute all the traffic units in an existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing the subsession options that are marked with an asterisk (*)

OPTION-> o /* The OLD LIST option is chosen */

Will you use the current traffic unit file (schreq.f#2)? (y/n) > n

Enter the name for the traffic unit file
It may be up to 13 characters long including the .ext.
Filenames may include only one '#' character
as the first character before the version number.

FILE NAME-> schreq.f#1

/* This is the resulting output */

(<NAME, Jud>, <GPA, 2>)
(<NAME, Steve>, <GPA, 1>)
(<NAME, Steve>, <GPA, 4>)

(<MAX(GPA),4>)

(<NAME, Steve>, <GPA, 1>)
(<NAME, Jud>, <GPA, 2>)
(<NAME, Steve>, <GPA, 4>)

/* Return to the Request-Interface Menu */

Enter the type of subsession you want
(s) SELECT; select traffic units from an existing list (or give new traffic units) for execution
(n) NEW LIST; create a new list of traffic units
(d) NEW DATABASE; choose a new database
(r) * REDIRECT OUTPUT; select output for answers
(p) * PERFORMANCE TESTING
(m) * MODIFY; modify an existing list of traffic units

(o) * OLD LIST; execute all the traffic units in an existing list
(x) EXIT; return to generate,load,execute, or exit menu

Please refer to the MLDS/MBDS user's manual before choosing the subsession options that are marked with an asterisk (*)

OPTION-> x /* Exit to the Attribute-Based Menu */

Welcome to the attribute-based/ABDL interface

(g) - generate a new database
(l) - load a new database
(r) - execute the request interface
(x) - exit to the previous system menu

OPTION-> x /* Exit to the System Menu */

Welcome to the MLDS/MBDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> x /* Exit from MLDS/MBDS */

This completes the second part of the manual, on using the attribute-based/ABDL interface of MBDS/MLDS.

## 3. USING THE MLDS LANGUAGE INTERFACE.

### 3.1. RELATIONAL/SQL INTERFACE

The relational/sql interface accepts input in the relational language syntax and translates it into the attribute-base data language for storage and retrieval. Create and request files can be read in either from a file or from the terminal. This manual will demonstrate both methods.

It should be noted, that if the terminal entry mode is chosen, the current database/create file will be lost if a new one is loaded and similarly with the request file, if the current file is modified/updated or a new file is loaded. In addition, when exiting the MBDS/MLDS system all information typed in will be lost. It is therefore advised that all files be created and modified/ updated utilizing the VI Editor in UNIX.

Under normal operation, for the insert, delete and update queries, only a completion statement would appear. For select and nested select queries, output is shown. If no output is shown, then the query had no qualifying records. However, for the benefit of the user, a translation of all queries into the attribute-base language will be shown.

The size/length of all database inputs are equivalent to those specified in the MBDS section of this manual.

/* The System Menu */

Welcome to the MLDS/MBDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> r /* The relational interface option is chosen */

/* The Load/Process Database Menu */

Enter type of operation desired
(l) - load new database
(p) - process existing database
(x) - return to the MLDS/MBDS system menu

Action --- > l

```
 /* The 'l' option is used to load a new database.
    You will be queried for a name and then how you
    desire to enter the database, either from a file
    or the terminal. */

Enter name of database ----> school
 /* Name of the new database. */

Enter mode of input desired
      (f) - read in a group of creates from a file
      (t) - read in creates from the terminal
      (x) - return to the main menu

Action --- > t  /* Terminal input mode is chosen. */

/* You will now create the equivalent of the MBDS
   template file. */

Please enter your transactions one at a time.
You may have multiple lines per transaction.
Each transaction must be separated by a line that
      only contains the character '@'.
After the last transaction, the last line must consist only
      of the '$' character to signal end-of-file.

Input the transactions on the following lines :

create table cors: cnum (char(10)),
                   plac (char(10)),
                   room (int(4))
@
create table stud: name (char(10)),
                   gpa (int(4)),
                   cnum (char(10))
$   /* End of create file input. */

/* The following questions equate to the MBDS descriptor
   file. You must decide whether each attribute will be
   an indexing attribute, and if so, what its type and
   value will be:
   option n equates to descriptor type c,
   option e equates to descriptor type b,
   option r equates to descriptor type a. */

The following are the Relations in the SCHOOL Database:
```

CORS
STUD

Beginning with the first Relation, we will present each Attribute of the relation. You will be prompted as to whether you wish to include that Attribute as an Indexing Attribute, and, if so, whether it is to be indexed based on strict EQUALITY, or based on a RANGE OF VALUES.

Strike RETURN when ready to continue.
Action --- >

Relation name: CORS
Attribute Name: CNUM

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n /* Not used as an indexing attribute (type c) */

Relation name: CORS
Attribute Name: PLAC

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > e /* Used as an indexing attribute (type b) */

Enter EQUALITY match value, or <CR> to exit:Span

Enter EQUALITY match value, or <CR> to exit:Root

Enter EQUALITY match value, or <CR> to exit:
/* A return to finish the input. */

Relation name: CORS
Attribute Name: ROOM

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n

Relation name: STUD
Attribute Name: NAME

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > r /* Used as an indexing attribute (type a) */

Enter Lower Bound, or <CR> to exit:Aaa

Enter Upper Bound:Mzz

Enter Lower Bound, or <CR> to exit:Naa

Enter Upper Bound:Zzz

Enter Lower Bound, or <CR> to exit:

Relation name: STUD
Attribute Name: GPA

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n

Relation name: STUD
Attribute Name: CNUM

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

```
Action --- > n

/* After loading a new database, control returns to the
   Load/Process Database Menu. */

Enter type of operation desired
      (l) - load new database
      (p) - process existing database
      (x) - return to the MLDS/MBDS system menu

Action --- > p
 /* The 'p' option is used to process the most recently
    entered database. You will be queried for the
    database name given under option 'l', how you would
    like to enter your queries, from a file or the
    terminal and an option to view the current database
    schema. */

Enter name of database ----> school
 /* Name of the previously loaded database. */

Enter mode of input desired
      (f) - read in a group of queries from a file
      (t) - read in queries from the terminal
      (d) - display the current database schema
      (x) - return to the previous menu

Action --- > d
 /* The 'd' option displays the current database schema,
    which equates to the MBDS template file. The
    differences include: the addition of the attribute
    name type length and a key value. The key value
    is set during the create table process but is not
    currently being utilized. It will be discussed in
    later versions of this manual. */

database name = SCHOOL, number of relations = 2
database type = RELATIONAL

relation_name = CORS, number of attributes = 3
      attr name = CNUM,   type = s,   length = 10,   key = FALSE
      attr name = PLAC,   type = s,   length = 10,   key = FALSE
      attr name = ROOM,   type = i,   length = 4,    key = FALSE

relation_name = STUD, number of attributes = 3
      attr name = NAME,   type = s,   length = 10,   key = FALSE
```

```
    attr name = GPA,     type = i,   length = 4,    key = FALSE
    attr name = CNUM,    type = s,   length = 10,   key = FALSE

Enter mode of input desired
    (f) - read in a group of queries from a file
    (t) - read in queries from the terminal
    (d) - display the current database schema
    (x) - return to the previous menu

Action --- > t  /* Terminal input mode is chosen. */

Please enter your transactions one at a time.
You may have multiple lines per transaction.
Each transaction must be separated by a line that
    only contains the character '@'.
After the last transaction, the last line must consist only
    of the '$' character to signal end-of-file.

Input the transactions on the following lines :

insert into cors (cnum, plac, room):
 <'c123', 'span', 332>
@
insert into cors (cnum, plac, room):
 <'c205', 'root', 117>
@
insert into stud (name, gpa, cnum):
 <'steve', 4, 'c123'>
@
insert into stud (name, gpa, cnum):
 <'jud', 3, 'c205'>
@
select cnum, plac, room
from cors
@
select name, gpa, cnum
from stud
@
select cnum, plac, room
from cors
where cnum in
 (select cnum
  from stud)
@
select cors.plac, cors.room, stud.name, stud.gpa
from cors, stud
```

```
where cors.cnum = stud.cnum
@
delete cors
where cnum = 'c123'
@
update stud
set gpa = 4
where cnum = 'c205'
$  /* End of the query file. */

/* Upon completion of the query inputs, your file is
  redisplayed with each query numbered sequentially. */

    1    insert into cors (cnum, plac, room):
          <'c123', 'span', 332>

    2    insert into cors (cnum, plac, room):
          <'c205', 'root', 117>

    3    insert into stud (name, gpa, cnum):
          <'steve', 4, 'c123'>

    4    insert into stud (name, gpa, cnum):
          <'jud', 3, 'c205'>

    5    select cnum, plac, room
         from cors

    6    select name, gpa, cnum
         from stud

    7    select cnum, plac, room
         from cors
         where cnum in
          (select cnum
           from stud)

    8    select cors.plac, cors.room, stud.name, stud.gpa
         from cors, stud
         where cors.cnum = stud.cnum

    9    delete cors
         where cnum = 'c123'

    10   update stud
         set gpa = 4
```

```
        where cnum = 'c205'

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)  - redisplay the file of queries
     (x)  - return to the previous menu

Action --- > 1  /* Execute the first insert. */

[ INSERT (<TEMP, Cors>, <CNUM, C123>, <PLAC, Span>, <ROOM, 332>) ]

Insert Query Done

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)  - redisplay the file of queries
     (x)  - return to the previous menu

Action --- > 2  /* Execute the second insert. */

[ INSERT (<TEMP, Cors>, <CNUM, C205>, <PLAC, Root>, <ROOM, 117>) ]

Insert Query Done

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)  - redisplay the file of queries
     (x)  - return to the previous menu

Action --- > 3  /* Execute the third insert. */

[ INSERT (<TEMP, Stud>, <NAME, Steve>, <GPA, 4>, <CNUM, C123>) ]

Insert Query Done

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)  - redisplay the file of queries
     (x)  - return to the previous menu

Action --- > 4  /* Execute the fourth insert. */

[ INSERT (<TEMP, Stud>, <NAME, Jud>, <GPA, 3>, <CNUM, C205>) ]

Insert Query Done
```

```
Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > 5  /* Execute a retrieve of the Cors table. */

[ RETRIEVE   (TEMP = Cors) (CNUM, PLAC, ROOM)  ]

CNUM      |PLAC      |ROOM |
-----------------------------------------
C205      |Root      |117     |
C123      |Span      |332     |

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > 6  /* Execute a retrieve of the Stud table. */

[ RETRIEVE   (TEMP = Stud) (NAME, GPA, CNUM)  ]

NAME      |GPA  |CNUM      |
-----------------------------------------
Jud       |3     |C205      |
Steve     |4     |C123      |

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > d  /* Redisplay the current file of queries. */

     1    insert into cors (cnum, plac, room):
          <'c123', 'span', 332>

     2    insert into cors (cnum, plac, room):
          <'c205', 'root', 117>

     3    insert into stud (name, gpa, cnum):
          <'steve', 4, 'c123'>

     4    insert into stud (name, gpa, cnum):
          <'jud', 3, 'c205'>
```

```
5    select cnum, plac, room
     from cors

6    select name, gpa, cnum
     from stud

7    select cnum, plac, room
     from cors
     where cnum in
      (select cnum
       from stud)

8    select cors.plac, cors.room, stud.name, stud.gpa
     from cors, stud
     where cors.cnum = stud.cnum

9    delete cors
     where cnum = 'c123'

10   update stud
     set gpa = 4
     where cnum = 'c205'
```

Pick the number or letter of the action desired
(num) - execute one of the preceding queries
(d) - redisplay the file of queries
(x) - return to the previous menu

Action --- > 7 /* Execute a nested retrieve. */

[ RETRIEVE ((TEMP = CORS) and (CNUM = **********))
(CNUM, PLAC, ROOM) ]
[ RETRIEVE (TEMP = Stud) (CNUM) ]
[ RETRIEVE ((TEMP = Cors) and (CNUM = C205)) or ((TEMP = Cors) and
(CNUM = C123)) (CNUM, PLAC, ROOM) ]

| CNUM | PLAC | ROOM |
|---|---|---|
| C205 | Root | 117 |
| C123 | Span | 332 |

Pick the number or letter of the action desired
(num) - execute one of the preceding queries
(d) - redisplay the file of queries
(x) - return to the previous menu

```
Action --- > 8  /* Execute a retrieve common. */

[ RETRIEVE   ( TEMP = Cors) (PLAC, ROOM)
  COMMON  (CNUM , CNUM)
  RETRIEVE   ( TEMP = Stud) (NAME, GPA)  ]

COMMON |PLAC      |ROOM |NAME |GPA |
-----------------------------------------------------------------
File         |Span      |332      |Steve   |4      |
File         |Root      |117      |Jud      |3      |

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > 9  /* Execute a delete. */

[ DELETE (  (TEMP = Cors) and (CNUM = C123) ) ]

Delete Query Done

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > 5  /* Execute a retrieve of the Cors table
                   to check the previous delete. */

[ RETRIEVE   (TEMP = Cors) (CNUM, PLAC, ROOM)  ]

CNUM      |PLAC      |ROOM |
-------------------------------------------
C205         |Root      |117      |

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > 10  /* Execute an update. */

[ UPDATE (  (TEMP = Stud) and (CNUM = C205) ) <GPA = 4> ]

Modify Query Done
```

```
Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > 6  /* Execute a retrieve of the Stud table
                   to check the previous update. */

[ RETRIEVE   (TEMP = Stud) (NAME, GPA, CNUM)  ]


NAME      |GPA  |CNUM      |
-----------------------------------------
Jud       |4     |C205      |
Steve     |4     |C123      |

Pick the number or letter of the action desired
     (num) - execute one of the preceding queries
     (d)   - redisplay the file of queries
     (x)   - return to the previous menu

Action --- > x  /* Exit from the Execution Menu. */

/* We have completed executing the current list of queries. */

Enter mode of input desired
     (f) - read in a group of queries from a file
     (t) - read in queries from the terminal
     (d) - display the current database schema
     (x) - return to the previous menu

Action --- > x  /* Exit to the Load/Process Database Menu. */

/* We now demonstrate the loading of a database and a list
   of queries from files. */

Enter type of operation desired
     (l) - load new database
     (p) - process existing database
     (x) - return to the MLDS/MBDS system menu

Action --- > 1  /* Load a new database option is chosen. */

Enter name of database ----> school2
  /* Name of the new database. */
```

Enter mode of input desired
(f) - read in a group of creates from a file
(t) - read in creates from the terminal
(x) - return to the main menu

Action --- > f /* Read from a file option is chosen. */

What is the name of the CREATE/QUERY file ----> sqldb
/* A UNIX file name. */

/*-We choose to have no indexing attributes for this database.
Option 'n' is chosen for all attributes. */

The following are the Relations in the SCHOOL2 Database:

COUR
PRER

Beginning with the first Relation, we will present each Attribute of the relation. You will be prompted as to whether you wish to include that Attribute as an Indexing Attribute, and, if so, whether it is to be indexed based on strict EQUALITY, or based on a RANGE OF VALUES.

Strike RETURN when ready to continue.
Action --- >

Relation name: COUR
Attribute Name: CNUM

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n

Relation name: COUR
Attribute Name: TITL

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n

Relation name: COUR
Attribute Name: DESC

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n

Relation name: PRER
Attribute Name: CNUM

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n

Relation name: PRER
Attribute Name: PNUM

Do you wish to install this Attribute as an Indexing Attribute?

(n) - no; continue with next Attribute/Relation
(e) - yes; establish this as an EQUALITY Attribute
(r) - yes; establish this as a RANGE Attribute

Action --- > n

/* The new database has been loaded, control returns to the
Load/Process Database Menu. */

Enter type of operation desired
(l) - load new database
(p) - process existing database
(x) - return to the MLDS/MBDS system menu

Action --- > p /* Process the current database option is chosen. */

Enter name of database ----> school2

```
 /* Name of the previously loaded database. */

Enter mode of input desired
     (f) - read in a group of queries from a file
     (t) - read in queries from the terminal
     (d) - display the current database schema
     (x) - return to the previous menu

Action --- > d  /* Display the schema option is chosen. */

database name = SCHOOL2, number of relations = 2
database type = RELATIONAL

relation_name = COUR, number of attributes = 3
     attr name = CNUM,   type = s,   length = 6,    key = FALSE
     attr name = TITL,   type = s,   length = 10,   key = FALSE
     attr name = DESC,   type = s,   length = 10,   key = FALSE

relation_name = PRER, number of attributes = 2
     attr name = CNUM,   type = s,   length = 6,    key = FALSE
     attr name = PNUM,   type = s,   length = 6,    key = FALSE

Enter mode of input desired
     (f) - read in a group of queries from a file
     (t) - read in queries from the terminal
     (d) - display the current database schema
     (x) - return to the previous menu

Action --- > f  /* Read queries from a file option is chosen. */

What is the name of the CREATE/QUERY file ----> sqlreqs
 /* A UNIX file name. */

     1    insert into cour (cnum, titl, desc) :
            <'c100', 'database', 'intro'>

     2    insert into cour (cnum, titl, desc) :
            <'c200', 'proglang', 'intro'>

     3    insert into cour (cnum, titl, desc) :
            <'c300', 'opersys', 'intro'>

     4    insert into prer (cnum, pnum) :
            <'c200', 'c100'>

     5    insert into prer (cnum, pnum) :
```

```
        <'c300', 'c200'>

    6   select cnum,titl,desc
        from cour

    7   select cnum, pnum
        from prer

    8   select cnum,titl,desc
        from cour
        where cnum in
          (select cnum
           from  prer)

    9   select cour.titl,cour.desc,prer.pnum
        from cour,prer
        where cour.cnum = prer.cnum

    10  delete cour
        where cnum='c100'

    11  update cour
        set titl = 'hello'
        where cnum = 'c300'

Pick the number or letter of the action desired
    (num) - execute one of the preceding queries
    (d)  - redisplay the file of queries
    (x)  - return to the previous menu

Action --- > x  /* Exit from the Execution Menu. */
 /* At this point, execution is the same as previously demonstrated. */

Enter mode of input desired
    (f) - read in a group of queries from a file
    (t) - read in queries from the terminal
    (d) - display the current database schema
    (x) - return to the previous menu

Action --- > x  /* Exit to the Load/Process Menu. */

Enter type of operation desired
    (l) - load new database
    (p) - process existing database
    (x) - return to the MLDS/MBDS system menu
```

Action --- > x /* Exit to the System Menu. */

Welcome to the MLDS/MBDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> x /* Exit from MLDS/MBDS. */

This completes the relational/sql interface part of the manual.

## 3.2. HIERARCHICAL/DL/I INTERFACE

The hierarchical/dl/1 interface is nearly identical in both menu selections and operations to the relational/sql interface. The major differences are:

(1) when loading a new database, it can only be read in from a file.
(2) there is no option to display the current database schema.
(3) when executing a request, the currency pointer must be reset to the root after each action unless retrieving more than one occurrence/record from the same parent (loop).

Some menu selections shown in detail in the previous section, will not be repeated here to reduce redundancy in this manual.

It is advised to utilize the VI Editor in UNIX to create and modify your files. Remember all data entered via the terminal mode is stored in dynamic memory and will be lost if the system fails or your datafile is changed.

Under normal operation, only completion messages appear after an insert, delete or replace request. For get unique and get next (looping), output is shown. NOTE: if you try to retrieve a non-existent occurrence/record the system will fail. However, a translation of all requests into the attribute-base language will be shown.

The size/length of all database inputs are equivalent to those specified in the MBDS section of this manual.

/* The System Menu. */

Welcome to the MLDS/MBDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> h /* The hierarchical interface option is chosen. */

/* The Load/Process Database Menu. */

Enter type of operation desired
(l) - load new database
(p) - process existing database
(x) - return to the operating system

Action --- > l /* The load a new database option is chosen. */

Enter name of database ----> ed5
/* This name must be the same as the one in your database file. */

Enter mode of input desired
(f) - read in database description from a file
(x) - return to the to main menu

Action --- > f /* Read the database from a file option is chosen. */

What is the name of the DBD/REQUEST file ----> dlidb5
/* A UNIX file name. */

/* The following database will be used for demonstrating the
hierarchical interface:

dbd name= ed5
segm name= course
field name= (csnum, seq), type= char, bytes= 10
field name= title, type= char, bytes= 10
field name= descripn, type= char, bytes= 10
segm name= prereq, parent= course
field name= (pcsnum, seq), type= char, bytes= 10
field name= title, type= char, bytes= 10
segm name= offering, parent= course

```
field  name= (date, seq), type= char, bytes= 10
field  name= location, type= char, bytes= 10
field  name= format, type= char, bytes= 10
$
*/

/* This section is identical to the one explained in the
   relational interface and will not be repeated. The only
   difference is the addition of the option 'n'. This option
   eliminates the displaying of all the segments/files and
   fields/attributes, if indexes are not desired for your
   database. */

The following are the Segments in the ED5 Database:

     COURSE
     PREREQ
     OFFERING

Beginning with the first Segment, we will present each
Field of that Segment.  You will be prompted as to whether
you wish to include that Field as an Indexing Field,
and, if so, whether it is to be indexed based on strict
EQUALITY, or based on a RANGE OF VALUES. If you do not want
to enter any indexes for your database, type an 'n' when
the Action --> prompt appears

Strike RETURN or 'n' when ready to continue.

Action --- > n /* No indexes desired for the database. */

/* After loading a new database, control returns to the Load/
   Process Database Menu. */

Enter type of operation desired
     (l) - load new database
     (p) - process existing database
     (x) - return to the operating system

Action --- > p /* The process an existing database option is chosen. */

Enter name of database ----> ed5
 /* The name of the previously loaded database. */

Enter mode of input desired
     (f) - read in a group of DL/I requests from a file
```

(t) - read in DL/I requests from the terminal
(x) - return to the previous menu

Action --- > f
/* The read from a file option is chosen.
The 't' option is identical to the one explained
in the relational interface and will not be
repeated. */

What is the name of the DBD/REQUEST file ----> dlireqs5
/* A UNIX file name. */

/* After entering the request file name, the file will be displayed
with each request numbered sequentially. The 'd' option in the
Execution Menu is identical to the one explained in the relational
interface and will not be repeated. */

```
1    build (csnum, title, descripn) :
       ('C100', 'database', 'intro')
     isrt course

2    build (csnum, title, descripn) :
       ('C200', 'proglang', 'intro')
     isrt course

3    build (csnum, title, descripn) :
       ('C300', 'opersys', 'intro')
     isrt course

4    build (date, location, format) : ('d1', 'monterey', 'mw')
     isrt course (csnum = 'c100')
          offering

5    build (date, location, format) : ('d2', 'monterey', 'mw')
     isrt course (csnum = 'c100')
          offering

6    build (date, location, format) : ('d3', 'monterey', 'mw')
     isrt course (csnum = 'c100')
          offering

7    gu course

8    aa gn course
          goto aa
```

```
9    gu course(csnum = 'c100')
       offering

10   bb gn offering
         goto bb

11   ghu course(csnum = 'c100')
       offering(date = 'd3')
     change location to 'washdc'
     repl

12   ghu course(csnum = 'c100')
       offering(date = 'd2')
     dlet
```

Pick the number or letter of the action desired
(num) - execute one of the preceding DL/I requests
(d) - redisplay the file of DL/I requests
(r) - reset the currency pointer to the root
(x) - return to the previous menu

Action --- > 1 /* Execute the first insert into the root. */

[INSERT (<TEMP, Course>, <CSNUM, C100>, <TITLE, Database>,
<DESCRIPN, Intro>)]

Insert accomplished

Pick the number or letter of the action desired
(num) - execute one of the preceding DL/I requests
(d) - redisplay the file of DL/I requests
(r) - reset the currency pointer to the root
(x) - return to the previous menu

Action --- > r /* Reset pointer to the root. */

Pick the number or letter of the action desired
(num) - execute one of the preceding DL/I requests
(d) - redisplay the file of DL/I requests
(r) - reset the currency pointer to the root
(x) - return to the previous menu

Action --- > 2 /* Execute the second insert into the root. */

[INSERT (<TEMP, Course>, <CSNUM, C200>, <TITLE, Proglang>,
<DESCRIPN, Intro>)]

```
Insert accomplished

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > 3 /* Execute the third insert into the root. */

[INSERT (<TEMP, Course>, <CSNUM, C300>, <TITLE, Opersys>,
 <DESCRIPN, Intro>)]

Insert accomplished

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > 4 /* Execute the first insert into a child. */

-- ISRT --
[RETRIEVE ((TEMP = COURSE) and (CSNUM = C100)) (CSNUM) BY CSNUM]

-- ISRT --
[INSERT (<TEMP, OFFERING>, <CSNUM, **********>, <DATE, D1>,
 <LOCATION, Monterey>, <FORMAT, Mw>)]
----- Parent is COURSE -----
```

```
[RETRIEVE ((TEMP = Course) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

[INSERT (<TEMP, Offering>, <CSNUM, C100>, <DATE, D1>,
 <LOCATION, Monterey>, <FORMAT, Mw>)]

Insert accomplished

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)   - redisplay the file of DL/I requests
     (r)   - reset the currency pointer to the root
     (x)   - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)   - redisplay the file of DL/I requests
     (r)   - reset the currency pointer to the root
     (x)   - return to the previous menu

Action --- > 5 /* Execute the second insert into a child. */

-- ISRT --
[RETRIEVE ((TEMP = COURSE) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

-- ISRT --
[INSERT (<TEMP, OFFERING>, <CSNUM, **********>, <DATE, D2>,
 <LOCATION, Monterey>, <FORMAT, Mw>)]
----- Parent is COURSE -----

[RETRIEVE ((TEMP = Course) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

[INSERT (<TEMP, Offering>, <CSNUM, C100>, <DATE, D2>,
 <LOCATION, Monterey>, <FORMAT, Mw>)]

Insert accomplished

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)   - redisplay the file of DL/I requests
     (r)   - reset the currency pointer to the root
     (x)   - return to the previous menu
```

```
Action --- > r

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > 6 /* Execute the third insert into a child. */

-- ISRT --
[RETRIEVE ((TEMP = COURSE) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

-- ISRT --
[INSERT (<TEMP, OFFERING>, <CSNUM, **********>, <DATE, D3>,
 <LOCATION, Monterey>, <FORMAT, Mw>)]
----- Parent is COURSE -----

[RETRIEVE ((TEMP = Course) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

[INSERT (<TEMP, Offering>, <CSNUM, C100>, <DATE, D3>,
 <LOCATION, Monterey>, <FORMAT, Mw>)]

Insert accomplished

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > 7 /* Execute a get unique from the root. */

[RETRIEVE (TEMP = Course) (CSNUM, TITLE, DESCRIPN) BY CSNUM]

CSNUM C100 TITLE Database DESCRIPN Intro
```

```
Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > 8
/* No Reset. Execute a get next from the root.
   Returns the rest of the occurrences in the root.*/

-- GN --
[RETRIEVE (TEMP = Course) (CSNUM, TITLE, DESCRIPN) BY CSNUM]
----- Loop to COURSE -----

CSNUM C200 TITLE Proglang DESCRIPN Intro

CSNUM C300 TITLE Opersys DESCRIPN Intro

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)  - redisplay the file of DL/I requests
     (r)  - reset the currency pointer to the root
     (x)  - return to the previous menu

Action --- > 9 /* Execute a get unique with a specific parent. */

-- GU --
[RETRIEVE ((TEMP = COURSE) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

-- GU --
[RETRIEVE ((TEMP = OFFERING) and (CSNUM = **********))
 (DATE, LOCATION, FORMAT) BY DATE]
----- Parent is COURSE -----

[RETRIEVE ((TEMP = Course) and (CSNUM = C100))
 (CSNUM) BY CSNUM]
```

```
[RETRIEVE ((TEMP = Offering) and (CSNUM = C100))
 (DATE, LOCATION, FORMAT) BY DATE]

DATE D1 LOCATION Monterey FORMAT Mw

Pick the number or letter of the action desired
      (num) - execute one of the preceding DL/I requests
      (d)   - redisplay the file of DL/I requests
      (r)   - reset the currency pointer to the root
      (x)   - return to the previous menu

Action --- > 10
/* No Reset. Executes a get next with the same parent.
   Returns the rest of the children. */

-- GN --
[RETRIEVE ((TEMP = OFFERING) and (CSNUM = **********))
 (DATE, LOCATION, FORMAT) BY DATE]
----- Loop to OFFERING -----

DATE D2 LOCATION Monterey FORMAT Mw

DATE D3 LOCATION Monterey FORMAT Mw

Pick the number or letter of the action desired
      (num) - execute one of the preceding DL/I requests
      (d)   - redisplay the file of DL/I requests
      (r)   - reset the currency pointer to the root
      (x)   - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
      (num) - execute one of the preceding DL/I requests
      (d)   - redisplay the file of DL/I requests
      (r)   - reset the currency pointer to the root
      (x)   - return to the previous menu

Action --- > 11
/* Execute a replace/update on a specific occurrence/record. */

-- GHU --
[RETRIEVE ((TEMP = COURSE) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

-- GHU --
```

```
[RETRIEVE ((TEMP = OFFERING) and (CSNUM = **********)
 and (DATE = D3)) (DATE) BY DATE]
----- Parent is COURSE -----

-- REPL --
[UPDATE ((TEMP = OFFERING) and (CSNUM = **********)
 and (DATE = **********)) <LOCATION = Washdc>]
----- Parent is OFFERING -----

[RETRIEVE ((TEMP = Course) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

[RETRIEVE ((TEMP = Offering) and (CSNUM = C100) and
 (DATE = D3)) (DATE) BY DATE]

[UPDATE ((TEMP = Offering) and (CSNUM = C100) and
 (DATE = D3)) <LOCATION = Washdc>]

Replace accomplished

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)   - redisplay the file of DL/I requests
     (r)   - reset the currency pointer to the root
     (x)   - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
     (num) - execute one of the preceding DL/I requests
     (d)   - redisplay the file of DL/I requests
     (r)   - reset the currency pointer to the root
     (x)   - return to the previous menu

Action --- > 12
/* Execute a delete on a specific occurrence/record. */

-- GHU --
[RETRIEVE ((TEMP = COURSE) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

-- GHU --
[RETRIEVE ((TEMP = OFFERING) and (CSNUM = **********) and
 (DATE = D2)) (DATE) BY DATE]
----- Parent is COURSE -----
```

```
-- DLET --
[DELETE ((TEMP = OFFERING) and (CSNUM = **********) and
 (DATE = **********))]
----- Parent is OFFERING -----

[RETRIEVE ((TEMP = Course) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

[RETRIEVE ((TEMP = Offering) and (CSNUM = C100) and
 (DATE = D2)) (DATE) BY DATE]

[DELETE ((TEMP = Offering) and (CSNUM = C100) and
 (DATE = D2))]

Dlet operation complete

Pick the number or letter of the action desired
      (num) - execute one of the preceding DL/I requests
      (d)   - redisplay the file of DL/I requests
      (r)   - reset the currency pointer to the root
      (x)   - return to the previous menu

Action --- > r

Pick the number or letter of the action desired
      (num) - execute one of the preceding DL/I requests
      (d)   - redisplay the file of DL/I requests
      (r)   - reset the currency pointer to the root
      (x)   - return to the previous menu

Action --- > 9
 /* Execute a get unique with a specific parent to set the path,
    for retrieving all the children, to check the previous
    replace and delete. */

-- GU --
[RETRIEVE ((TEMP = COURSE) and (CSNUM = C100))
 (CSNUM) BY CSNUM]

-- GU --
[RETRIEVE ((TEMP = OFFERING) and (CSNUM = **********))
 (DATE, LOCATION, FORMAT) BY DATE]
----- Parent is COURSE -----

[RETRIEVE ((TEMP = Course) and (CSNUM = C100))
 (CSNUM) BY CSNUM]
```

```
[RETRIEVE ((TEMP = Offering) and (CSNUM = C100))
 (DATE, LOCATION, FORMAT) BY DATE]

DATE D1 LOCATION Monterey FORMAT Mw

Pick the number or letter of the action desired
      (num) - execute one of the preceding DL/I requests
      (d)   - redisplay the file of DL/I requests
      (r)   - reset the currency pointer to the root
      (x)   - return to the previous menu

Action --- > 10
/* No Reset. Execute a get unique with the same parent to
   check the previous replace and delete. */

-- GN --
[RETRIEVE ((TEMP = OFFERING) and (CSNUM = **********))
 (DATE, LOCATION, FORMAT) BY DATE]
----- Loop to OFFERING -----

DATE D3 LOCATION Washdc FORMAT Mw

Pick the number or letter of the action desired
      (num) - execute one of the preceding DL/I requests
      (d)   - redisplay the file of DL/I requests
      (r)   - reset the currency pointer to the root
      (x)   - return to the previous menu

Action --- > x /* Exit from the Execution Menu. */

Enter mode of input desired
      (f) - read in a group of DL/I requests from a file
      (t) - read in DL/I requests from the terminal
      (x) - return to the previous menu

Action --- > x /* Exit to the Load/Process Menu. */

Enter type of operation desired
      (l) - load new database
      (p) - process existing database
      (x) - return to the operating system

Action --- > x /* Exit to the System Menu. */

Welcome to the MLDS/MBDS Database System.
```

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> x /* Exit from MLDS/MBDS. */

This completes the hierarchical/dl/1 interface part of the manual.

## 3.3. NETWORK/CODASYL INTERFACE

The network/CODASYL interface is similar in both menu selections and operations to the relational and hierarchical interfaces. A few differences in menu selections between network and the other interfaces are:

(1) a new database can only be read in from a file. (network and hierarchical only)
(2) when queried about utilizing indexing attributes, an option 'n' is available. This eliminates the displaying of all record and attribute names, if indexes are not desired for your database. (network and hierarchical only)
(3) there is an option to display the current database schema. (network and relational only)

All other menu selections are identical.

Some menu selections shown in detail in a previous section, will not be repeated here.

This implementation of CODASYL does not support a looping construct. Therefore, each set of attribute values in a record must be individually retrieved (get operation). Due to this limitation, a get operation can not be performed on a previously deleted (erased) set of attribute values because any reference to a non-existent set will cause a system failure.

It is recommended that the VI Editor in UNIX be utilized to create and modify your files, because all data entered via the terminal mode is stored in dynamic memory.

There are no completion statements following the store (insert), modify (update), and erase (delete) queries. However, a translation of all requests, into the attribute-base language, will be shown.

The size/length of all database inputs are equivalent to those specified in the MBDS section of this manual.

/* The System Menu. */

Welcome to the MLDS/MBDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> n /* The network interface option is chosen. */

/* The Load/Process Database Menu. */

Enter type of operation desired
(l) - load new database
(p) - process existing database
(x) - return to the operating system

Action --- > l /* The load a new database option is chosen. */

Enter name of database ----> sps
/* This name must be the same as the one in your database file. */

Enter mode of input desired
(f) - read in database description from a file
(x) - return to the to main menu

Action --- > f /* Read the database from a file option is chosen. */

What is the name of the DBD/REQUEST file ----> dmldb
/* A UNIX file name. */

/* The following database will be used for demonstrating the
network interface:

SCHEMA NAME IS SPS;
RECORD NAME IS SA;
DUPLICATES ARE NOT ALLOWED FOR SNO;
SNO ; CHARACTER 10.

```
   SNAME ;  CHARACTER 10.
   CITY ;  CHARACTER 10.
RECORD NAME IS PA;
 DUPLICATES ARE NOT ALLOWED FOR PNO;
   PNO ;  CHARACTER 10.
   PNAME ;  CHARACTER 10.
   CITY ;  CHARACTER 10.
RECORD NAME IS SP;
 DUPLICATES ARE NOT ALLOWED FOR SNO,PNO;
   SNO ;  CHARACTER 10.
   PNO ;  CHARACTER 10.
   QTY ;  FIXED 4.
SET NAME IS SSP;
 OWNER IS SA;
 MEMBER IS SP;
   INSERTION IS AUTOMATIC
   RETENTION IS FIXED;
   SET SELECTION IS BY VALUE OF SNO IN SA;
SET NAME IS PSP;
 OWNER IS PA;
 MEMBER IS SP;
   INSERTION IS AUTOMATIC
   RETENTION IS FIXED;
   SET SELECTION IS BY VALUE OF PNO IN PA;
$
*/

/* This section is identical to the one explained in the
   previous interfaces and will not be repeated. */
```

The following are the Records in the SPS Database:

SA
PA
SP

Beginning with the first Record, we will present each Attribute of that Record. You will be prompted as to whether you wish to include that Attribute as an Indexing Attribute, and, if so, whether it is to be indexed based on strict EQUALITY, or based on a RANGE OF VALUES. If you do not want to enter any indexes for your database, type an 'n' when the Action --> prompt appears

Strike RETURN or 'n' when ready to continue.

```
Action --- > n /* No indexes desired for the database. */

/* After loading a new database, control returns to the
  Load/Process Database Menu. */

Enter type of operation desired
      (l) - load new database
      (p) - process existing database
      (x) - return to the operating system

Action --- > p /* The process an existing database option is chosen. */

Enter name of database ----> sps
/* The name of the previously loaded database. */

Enter mode of input desired
      (f) - read in a group of CODASYL requests from a file
      (t) - read in CODASYL requests from the terminal
      (d) - display the current database schema
      (x) - return to the previous menu

Action --- > d
/* The 'd' option displays the current database schema, listing
   each record and its attributes. In addition, the set name is
   listed along with the owner and member names. */

The record name is : SA
The attributes of the record are: SNO SNAME CITY

The record name is : PA
The attributes of the record are: PNO PNAME CITY

The record name is : SP
The attributes of the record are: SNO PNO QTY

The set name is : SSP
The owner of the set is: SA
The member of the set is: SP

The set name is : PSP
The owner of the set is: PA
The member of the set is: SP

Enter mode of input desired
      (f) - read in a group of CODASYL requests from a file
      (t) - read in CODASYL requests from the terminal
```

(d) - display the current database schema
(x) - return to the previous menu

Action --- > f
/* The read from a file option is chosen. The 't' option
is identical to the one explained in the relational
interface and will not be repeated. */

What is the name of the DBD/REQUEST file ----> dmlreqs
/* A UNIX file name. */

/* After entering the request file name, the file will be
displayed with each request numbered sequentially.
The 'd' option in the Execution Menu is identical to
the one explained in the relational interface and will
not be repeated. */

```
1   MOVE SS1 TO SNO IN SA
    MOVE DEC TO SNAME IN SA
    MOVE MONT TO CITY IN SA
    STORE SA

2   MOVE SS2 TO SNO IN SA
    MOVE IBM TO SNAME IN SA
    MOVE SANJ TO CITY IN SA
    STORE SA

3   MOVE PP1 TO PNO IN PA
    MOVE NUT TO PNAME IN PA
    MOVE MONT TO CITY IN PA
    STORE PA

4   MOVE PP2 TO PNO IN PA
    MOVE BOLT TO PNAME IN PA
    MOVE SANJ TO CITY IN PA
    STORE PA

5   MOVE SS1 TO SNO IN SA
    MOVE PP1 TO PNO IN PA
    MOVE SS1 TO SNO IN SP
    MOVE PP1 TO PNO IN SP
    MOVE 100 TO QTY IN SP
    STORE SP

6   MOVE SS1 TO SNO IN SA
    FIND ANY SA USING SNO IN SA
```

```
        GET SA

    7   MOVE SS1 TO SNO IN SP
        FIND ANY SP USING SNO IN SP
        GET SP

    8   MOVE SS1 TO SNO IN SP
        MOVE PP1 TO PNO IN SP
        MOVE 300 TO QTY IN SP
        MODIFY SP

    9   MOVE SS1 TO SNO IN SA
        MOVE SONY TO SNAME IN SA
        MOVE TOKYO TO CITY IN SA
        FIND ANY SA USING SNO IN SA
        MODIFY SNAME, CITY IN SA

    10  MOVE SS2 TO SNO IN SA
        FIND ANY SA USING SNO IN SA
        ERASE SA

Pick the number or letter of the action desired
     (num) - execute one of the preceding CODASYL requests
     (d)  - redisplay the file of CODASYL requests
     (x)  - return to the previous menu

Action --- > 1
/* Execute the first store into the first owner. */

[RETRIEVE (((TEMP = SA) and (SNO = SS1))) (DBKEY) BY DBKEY]
[INSERT (<TEMP,SA>,<DBKEY,******>,<SNO,SS1>,<SNAME,DEC>,
<CITY,MONT>)]

[INSERT (<TEMP,Sa>,<DBKEY,1>,<SNO,Ss1>,<SNAME,Dec>,<CITY,Mont>)]

Pick the number or letter of the action desired
     (num) - execute one of the preceding CODASYL requests
     (d)  - redisplay the file of CODASYL requests
     (x)  - return to the previous menu

Action --- > 2
/* Execute the second store into the first owner. */

[RETRIEVE (((TEMP = SA) and (SNO = SS2))) (DBKEY) BY DBKEY]
[INSERT (<TEMP,SA>,<DBKEY,******>,<SNO,SS2>,<SNAME,IBM>,
<CITY,SANJ>)]
```

[INSERT (<TEMP,Sa>,<DBKEY,2>,<SNO,Ss2>,<SNAME,Ibm>,<CITY,Sanj>)]

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 3
/* Execute the first store into the second owner. */

[RETRIEVE (((TEMP = PA) and (PNO = PP1))) (DBKEY) BY DBKEY]
[INSERT (<TEMP,PA>,<DBKEY,******>,<PNO,PP1>,<PNAME,NUT>,
<CITY,MONT>)]

[INSERT (<TEMP,Pa>,<DBKEY,3>,<PNO,Pp1>,<PNAME,Nut>,<CITY,Mont>)]

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 4
/* Execute the second store into the second owner. */

[RETRIEVE (((TEMP = PA) and (PNO = PP2))) (DBKEY) BY DBKEY]
[INSERT (<TEMP,PA>,<DBKEY,******>,<PNO,PP2>,<PNAME,BOLT>,
<CITY,SANJ>)]

[INSERT (<TEMP,Pa>,<DBKEY,4>,<PNO,Pp2>,<PNAME,Bolt>,<CITY,Sanj>)]

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 5
/* Execute a store into the member. */

[RETRIEVE (((TEMP = SP) and (SNO = SS1))or((TEMP = SP) and (PNO = PP1)))
(DBKEY) BY DBKEY]
[RETRIEVE ((TEMP = SA) and (SNO = SS1)) (DBKEY)]

[RETRIEVE ((TEMP = PA) and (PNO = PP1)) (DBKEY)]

[INSERT (<TEMP,SP>,<DBKEY,******>,<SNO,SS1>,<PNO,PP1>,<QTY,100>,
<MEMSSP,******>,<MEMPSP,******>)]

```
[RETRIEVE ((TEMP = Sp) and (SNO = Ss1))or((TEMP = Sp) and (PNO = Pp1))
 (DBKEY) BY DBKEY]

[RETRIEVE ((TEMP = Sa) and (SNO = Ss1)) (DBKEY)]

[RETRIEVE ((TEMP = Pa) and (PNO = Pp1)) (DBKEY)]

[INSERT (<TEMP,Sp>,<DBKEY,5>,<SNO,Ss1>,<PNO,Pp1>,<QTY,100>,
 <MEMSSP,1>, <MEMPSP,3>)]

Pick the number or letter of the action desired
      (num) - execute one of the preceding CODASYL requests
      (d)   - redisplay the file of CODASYL requests
      (x)   - return to the previous menu

Action --- > 6
/* Execute a get on the first owner. */

[RETRIEVE ((TEMP = SA) and (SNO = SS1)) (SNO, SNAME, CITY, DBKEY)
 BY DBKEY]

[RETRIEVE ((TEMP = Sa) and (SNO = Ss1)) (SNO, SNAME, CITY, DBKEY)
 BY DBKEY]

SNO Ss1 SNAME Dec CITY Mont

Pick the number or letter of the action desired
      (num) - execute one of the preceding CODASYL requests
      (d)   - redisplay the file of CODASYL requests
      (x)   - return to the previous menu

Action --- > 9
/* Execute a modify on a set of attribute values in the first
   owner. */

[RETRIEVE ((TEMP = SA) and (SNO = SS1)) (SNO, SNAME, CITY, DBKEY)
 BY DBKEY]

[RETRIEVE ((TEMP = Sa) and (SNO = Ss1)) (SNO, SNAME, CITY, DBKEY)
 BY DBKEY]

SA
SA
SNAME_^^^^
CITY_^^^^
```

[UPDATE ((TEMP = SA) and (DBKEY = 1)) <SNAME = SONY>]

[UPDATE ((TEMP = SA) and (DBKEY = 1)) <CITY = TOKYO>]

[UPDATE ((TEMP = Sa) and (DBKEY = 1)) <SNAME = Sony>]

[UPDATE ((TEMP = Sa) and (DBKEY = 1)) <CITY = Tokyo>]

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 6
/* Execute a get to check the previously modified set of attribute values. */

[RETRIEVE ((TEMP = SA) and (SNO = SS1)) (SNO, SNAME, CITY, DBKEY) BY DBKEY]

[RETRIEVE ((TEMP = Sa) and (SNO = Ss1)) (SNO, SNAME, CITY, DBKEY) BY DBKEY]

SNO Ss1 SNAME Sony CITY Tokyo

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 7
/* Execute a get on the member. */

[RETRIEVE ((TEMP = SP) and (SNO = SS1)) (SNO, PNO, QTY, MEMSSP, MEMPSP, DBKEY) BY DBKEY]

[RETRIEVE ((TEMP = Sp) and (SNO = Ss1)) (SNO, PNO, QTY, MEMSSP, MEMPSP, DBKEY) BY DBKEY]

SNO Ss1 PNO Pp1 QTY 100

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 8
/* Execute a modify on a set of attribute values in the member. */

SNO_ &&&&&
PNO_ &&&&&
QTY_ &&&&&

[UPDATE ((TEMP = SP) and (DBKEY = 5)) <SNO = SS1>]

[UPDATE ((TEMP = SP) and (DBKEY = 5)) <PNO = PP1>]

[UPDATE ((TEMP = SP) and (DBKEY = 5)) <QTY = 300>]

[UPDATE ((TEMP = Sp) and (DBKEY = 5)) <SNO = Ss1>]

[UPDATE ((TEMP = Sp) and (DBKEY = 5)) <PNO = Pp1>]

[UPDATE ((TEMP = Sp) and (DBKEY = 5)) <QTY = 300>]

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 7
/* Execute a get to check the previously modified set of attribute values. */

[RETRIEVE ((TEMP = SP) and (SNO = SS1)) (SNO, PNO, QTY, MEMSSP,
MEMPSP, DBKEY) BY DBKEY]

[RETRIEVE ((TEMP = Sp) and (SNO = Ss1)) (SNO, PNO, QTY, MEMSSP,
MEMPSP, DBKEY) BY DBKEY]

SNO Ss1 PNO Pp1 QTY 300

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > 10
/* Execute an erase on a set of attribute values in the first owner. */

[RETRIEVE ((TEMP = SA) and (SNO = SS2)) (SNO, SNAME, CITY, DBKEY)
BY DBKEY]

[RETRIEVE ((TEMP = Sa) and (SNO = Ss2)) (SNO, SNAME, CITY, DBKEY) BY DBKEY]

[DELETE ((TEMP = SA) and (DBKEY = 2))]

[RETRIEVE ((MEMSSP = 2)) (DBKEY) BY DBKEY]

[DELETE ((TEMP = Sa) and (DBKEY = 2))]

Pick the number or letter of the action desired
(num) - execute one of the preceding CODASYL requests
(d) - redisplay the file of CODASYL requests
(x) - return to the previous menu

Action --- > x /* Exit from the Execution Menu. */

Enter mode of input desired
(f) - read in a group of CODASYL requests from a file
(t) - read in CODASYL requests from the terminal
(d) - display the current database schema
(x) - return to the previous menu

Action --- > x /* Exit to the Load/Process Menu. */

Enter type of operation desired
(l) - load new database
(p) - process existing database
(x) - return to the operating system

Action --- > x /* Exit to the System Menu. */

Welcome to the MLDS/MBDS Database System.

What operation would you like to perform?

(a) - execute the attribute-based/ABDL interface
(r) - execute the relational/SQL interface
(h) - execute the hierarchical/DL/I interface
(n) - execute the network/CODASYL interface
(f) - execute the functional/DAPLEX interface
(x) - exit the MLDS/MBDS system

OPTION-> x /* Exit from MLDS/MBDS. */

This completes the network/CODASYL interface part of the manual.

## LIST OF REFERENCES


[1] Demurjian, S. A. and Hsiao, D. K., "New Directions in Database-Systems Research and Development," Technical Report, NPS-52-85-001, Naval Postgraduate School, Monterey, California, February 1985.

[2] Demurjian, S. A. and Hsiao, D. K., "A Multi-Backend Database System for Performance Gains, Capacity Growth and Hardware Upgrade," *Proceedings of the 1986 2nd International Conference on Data Engineering* (February 1986).

[3] Hsiao, D. K. and Menon, M. J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part I)," Technical Report, OSU-CISRC-TR-81-7, The Ohio State University, Columbus, Ohio, July 1981.

[4] Hsiao, D. K. and Menon, M. J., "Design and Analysis of a Multi-Backend Database System for Performance Improvement, Functionality Expansion and Capacity Growth (Part II)," Technical Report, OSU-CISRC-TR-81-8, The Ohio State University, Columbus, Ohio, August 1981.

[5] Demurjian, S. A. and Hsiao, D. K., "The Multi-Lingual Database System," *Proceedings of the 1987 3rd International Conference on Data Engineering* (February 1987).

INITIAL DISTRIBUTION LIST

| | | No. Copies |
|---|---|---|
| 1. | Defense Technical Information Center<br>Cameron Station<br>Alexandria, Virginia 22304-6145 | 2 |
| 2. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93943-5002 | 2 |
| 4. | Department Chairman, Code 52<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 2 |
| 5. | Curriculum Officer, Code 37<br>Computer Technology<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 1 |
| 6. | Professor David K. Hsiao, Code 52Hq<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, California 93943-5000 | 2 |
| 7. | Professor Steven A. Demurjian<br>Computer Science & Engineering Department<br>The University of Connecticut<br>260 Glenbrook Road<br>Storrs, Connecticut 06268 | 2 |
| 8. | Col Myron W. Little<br>608 Claymont Drive<br>Ballwin, Missouri 63011 | 1 |
| 9. | Lt Craig W. Little<br>105 Kingston Ave<br>Kernersville, North Carolina 27284 | 2 |

Little
The design and implementation of pedagogical software for multi-back-end/multi-lingual database system.